3-3-6：每月巡检-统计邮件平台环境报告并输出HTML

作者简介

曾垂鑫，毕业于中国人民大学商务管理专业。微软（2013-2017五届）MVP，具有丰富的项目实施和解决方案经验，51CTO传媒推荐博客、博客之星，微软MCP/MCTS/MCITP/MCSA/MCSE认证系统专家和解决方案顾问。
属于实战派的讲师，具备丰富的万人规模以上企业的运维经验。课程内容侧重于实战实用。

曾垂鑫的博客地址：http://543925535.blog.51cto.com/
曾垂鑫的课程地址：http://edu.51cto.com/lecturer/index/user_id-639838.html

课程后续会形成完善的Windows运维工程师路线图，涵盖企业常用的Windows平台运维技术，后期课程包括但不限于AD、Exchange、Windows、System Center、Powershell、Hyper-v、Office 365等。

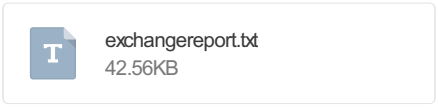
场景

查看整个环境的具体情况，每个月可以输出一次。

脚本下载地址：

Generate Exchange Environment Reports using Powershell 示例
https://gallery.technet.microsoft.com/exchange/Generate-Exchange-2388e7c9

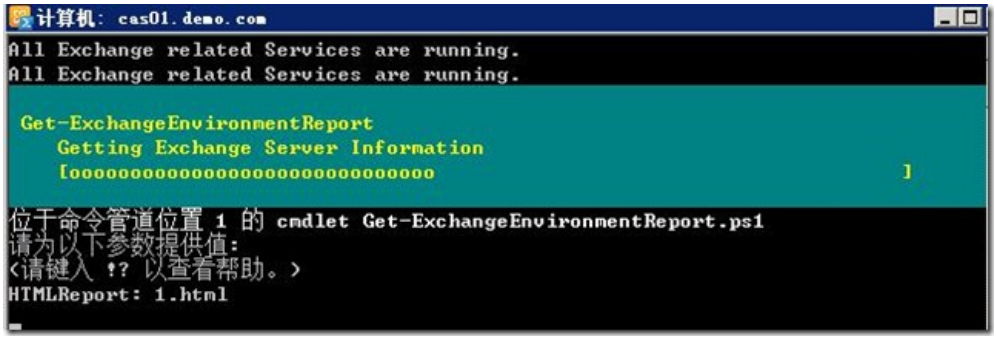
脚本来自于脚本中心，非常好用，可以获取到我们当前exchange环境的配置情况。
比如：服务器的数量、数据库的数量、用户数量、服务器的版本、操作系统的版本、邮箱大小等。
下载地址如下：
Generate Exchange Environment Reports using Powershell 示例
https://gallery.technet.microsoft.com/exchange/Generate-Exchange-2388e7c9
这个脚本非常易于使用，不需要做任何修改，直接在powershell中运行即可。
脚本是带param的，所以后面可以指定参数，如图。




```
param(
    [parameter(Position=0,Mandatory=$true,ValueFromPipeline=$false,HelpMessage='Filename to
write HTML report to'))][string]$HTMLReport,
        [parameter(Position=1,Mandatory=$false,ValueFromPipeline=$false,HelpMessage='Send
Mail ($True/$False)'))][bool]$SendMail=$false,
        [parameter(Position=2,Mandatory=$false,ValueFromPipeline=$false,HelpMessage='Mail
From'))][string]$MailFrom,
        [parameter(Position=3,Mandatory=$false,ValueFromPipeline=$false,HelpMessage='Mail
To'))]$MailTo,
        [parameter(Position=4,Mandatory=$false,ValueFromPipeline=$false,HelpMessage='Mail
Server'))][string]$MailServer,
        [parameter(Position=4,Mandatory=$false,ValueFromPipeline=
$false,HelpMessage='Schedule as user'))][string]$ScheduleAs,
        [parameter(Position=5,Mandatory=$false,ValueFromPipeline=$false,HelpMessage='Change
view to entire forest'))][bool]$ViewEntireForest=$true,
        [parameter(Position=5,Mandatory=$false,ValueFromPipeline=$false,HelpMessage='Server
Name Filter (eg NL-*)'))][string]$ServerFilter="*"
    )
```

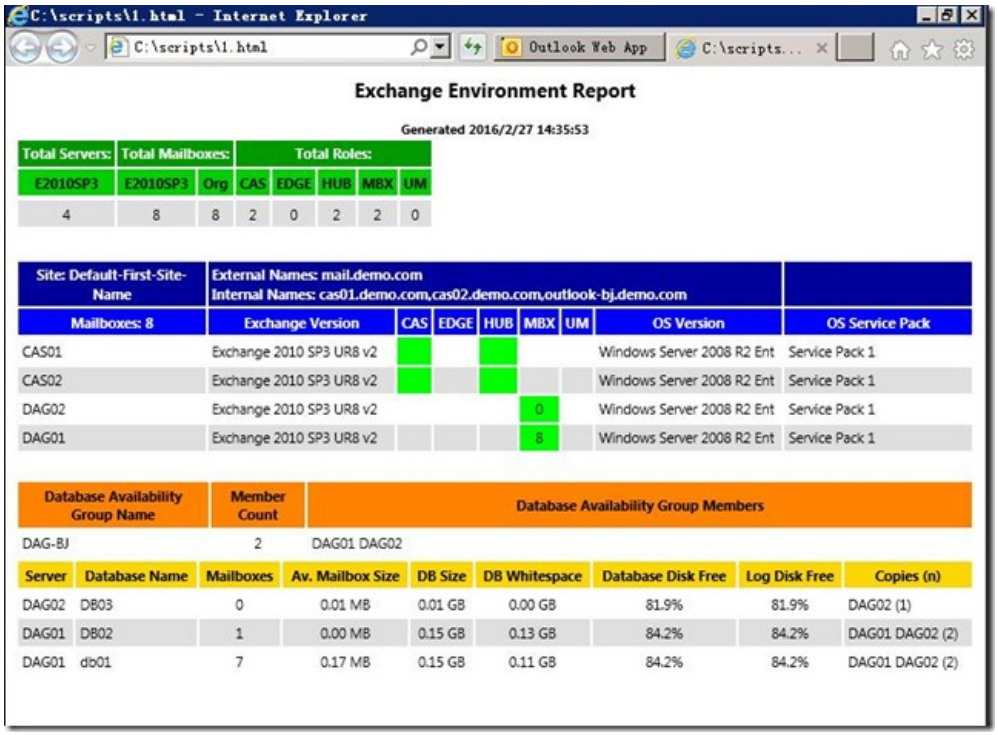
c7ff88ddbe4342b69decb15a0dc51e48

下图我输入了要保存为html的报告的名称，在这个脚本中HTMLReport是强制参数，必须提供值的。



fded7965f64844eead35a3cc6c51fa93

跑完了之后，生成的html如图所示。



7f531c899fca47ebb1f491294dc54731

改过后的脚本如下，整个拷贝到TXT中，重命名为PS1格式。

```
<#
  .SYNOPSIS
  Creates a HTML Report describing the Exchange environment

  Steve Goodman

THIS CODE IS MADE AVAILABLE AS IS, WITHOUT WARRANTY OF ANY KIND. THE ENTIRE
RISK OF THE USE OR THE RESULTS FROM THE USE OF THIS CODE REMAINS WITH THE USER.

Version 1.6.1 September 2015

  .DESCRIPTION

  This script creates a HTML report showing the following information about an Exchange
  2016, 2013, 2010 and to a lesser extent, 2007 and 2003, environment.

  The following is shown:

* Report Generation Time
* Total Servers per Exchange Version (2003 > 2010 or 2007 > 2016)
* Total Mailboxes per Exchange Version, Office 365 and Organisation
```

* Total Roles in the environment

Then, per site:
* Total Mailboxes per site
  * Internal, External and CAS Array Hostnames
* Exchange Servers with:
 o Exchange Server Version
 o Service Pack
 o Update Rollup and rollup version
 o Roles installed on server and mailbox counts
 o OS Version and Service Pack

Then, per Database availability group (Exchange 2010/2013/2016):
* Total members per DAG
* Member list
* Databases, detailing:
 o Mailbox Count and Average Size
 o Archive Mailbox Count and Average Size (Only shown if DAG includes Archive Mailboxes)
 o Database Size and whitespace
 o Database and log disk free
 o Last Full Backup (Only shown if one or more DAG database has been backed up)
 o Circular Logging Enabled (Only shown if one or more DAG database has Circular Logging enabled)
 o Mailbox server hosting active copy
 o List of mailbox servers hosting copies and number of copies

Finally, per Database (Non DAG DBs/Exchange 2007/Exchange 2003)
* Databases, detailing:
 o Storage Group (if applicable) and DB name
 o Server hosting database
 o Mailbox Count and Average Size
 o Archive Mailbox Count and Average Size (Only shown if DAG includes Archive Mailboxes)
 o Database Size and whitespace
 o Database and log disk free
 o Last Full Backup (Only shown if one or more DAG database has been backed up)
 o Circular Logging Enabled (Only shown if one or more DAG database has Circular Logging enabled)

This does not detail public folder infrastructure, or examine Exchange 2007/2003 CCR/SCC clusters
(although it attempts to detect Clustered Exchange 2007/2003 servers, signified by ClusMBX).

IMPORTANT NOTE: The script requires WMI and Remote Registry access to Exchange servers from the server
it is run from to determine OS version, Update Rollup, Exchange 2007/2003 cluster and DB size information.

.PARAMETER HTMLReport
  Filename to write HTML Report to

.PARAMETER SendMail
Send Mail after completion. Set to $True to enable. If enabled, -MailFrom, -MailTo, -MailServer are mandatory

.PARAMETER MailFrom
Email address to send from. Passed directly to Send-MailMessage as -From

.PARAMETER MailTo

```
 Email address to send to. Passed directly to Send-MailMessage as -To

 .PARAMETER MailServer
 SMTP Mail server to attempt to send through. Passed directly to Send-MailMessage as -SmtpServer

 .PARAMETER ScheduleAs
 Attempt to schedule the command just executed for 10PM nightly. Specify the username here, schtasks (under the hood) will
ask for a password later.

 .PARAMETER ViewEntireForest
 By default, true. Set the option in Exchange 2007 or 2010 to view all Exchange servers and recipients in the forest.

   .PARAMETER ServerFilter
 Use a text based string to filter Exchange Servers by, e.g. NL-* -  Note the use of the wildcard (*) character to allow for
multiple matches.

 .EXAMPLE
   Generate the HTML report
   .\Get-ExchangeEnvironmentReport.ps1 -HTMLReport .\report.html

   #>

param(
   [parameter(Position=0,Mandatory=$false,ValueFromPipeline=$false,HelpMessage='Filename to write HTML report to')]
[string]$HTMLReport="C:\PS\2017.html",
 [parameter(Position=1,Mandatory=$false,ValueFromPipeline=$false,HelpMessage='Send Mail ($True/$False)')]
[bool]$SendMail=$false,
 [parameter(Position=2,Mandatory=$false,ValueFromPipeline=$false,HelpMessage='Mail From')][string]$MailFrom,
 [parameter(Position=3,Mandatory=$false,ValueFromPipeline=$false,HelpMessage='Mail To')]$MailTo,
 [parameter(Position=4,Mandatory=$false,ValueFromPipeline=$false,HelpMessage='Mail Server')][string]$MailServer,
 [parameter(Position=4,Mandatory=$false,ValueFromPipeline=$false,HelpMessage='Schedule as user')][string]$ScheduleAs,
 [parameter(Position=5,Mandatory=$false,ValueFromPipeline=$false,HelpMessage='Change view to entire forest')]
[bool]$ViewEntireForest=$true,
 [parameter(Position=5,Mandatory=$false,ValueFromPipeline=$false,HelpMessage='Server Name Filter (eg NL-*)')]
[string]$ServerFilter="*"
   )

# Sub-Function to Get Database Information. Shorter than expected..
function _GetDAG
{
 param($DAG)
 @{Name   = $DAG.Name.ToUpper()
  MemberCount = $DAG.Servers.Count
  Members  = [array]($DAG.Servers | % { $_.Name })
  Databases  = @()
  }
}


# Sub-Function to Get Database Information
function _GetDB
{
```

```
param($Database,$ExchangeEnvironment,$Mailboxes,$ArchiveMailboxes,$E2010)

# Circular Logging, Last Full Backup
if ($Database.CircularLoggingEnabled) { $CircularLoggingEnabled="Yes" } else { $CircularLoggingEnabled = "No" }
if ($Database.LastFullBackup) { $LastFullBackup=$Database.LastFullBackup.ToString() } else { $LastFullBackup = "Not
Available" }

# Mailbox Average Sizes
$MailboxStatistics = [array]($ExchangeEnvironment.Servers[$Database.Server.Name].MailboxStatistics | Where {$_.Database -
eq $Database.Identity})
if ($MailboxStatistics)
{
 [long]$MailboxItemSizeB = 0
 $MailboxStatistics | %{ $MailboxItemSizeB+=$_.TotalItemSizeB }
 [long]$MailboxAverageSize = $MailboxItemSizeB / $MailboxStatistics.Count
} else {
 $MailboxAverageSize = 0
}

# Free Disk Space Percentage
if ($ExchangeEnvironment.Servers[$Database.Server.Name].Disks)
{
 foreach ($Disk in $ExchangeEnvironment.Servers[$Database.Server.Name].Disks)
 {
  if ($Database.EdbFilePath.PathName -like "$($Disk.Name)*")
  {
   $FreeDatabaseDiskSpace = $Disk.FreeSpace / $Disk.Capacity * 100
  }
  if ($Database.ExchangeVersion.ExchangeBuild.Major -ge 14)
  {
   if ($Database.LogFolderPath.PathName -like "$($Disk.Name)*")
   {
    $FreeLogDiskSpace = $Disk.FreeSpace / $Disk.Capacity * 100
   }
  } else {
   $StorageGroupDN = $Database.DistinguishedName.Replace("CN=$($Database.Name),","")
   $Adsi=[adsi]"LDAP://$($Database.OriginatingServer)/$($StorageGroupDN)"
   if ($Adsi.msExchESEParamLogFilePath -like "$($Disk.Name)*")
   {
    $FreeLogDiskSpace = $Disk.FreeSpace / $Disk.Capacity * 100
   }
  }
 }
} else {
 $FreeLogDiskSpace=$null
 $FreeDatabaseDiskSpace=$null
}

if ($Database.ExchangeVersion.ExchangeBuild.Major -ge 14 -and $E2010)
{
 # Exchange 2010 Database Only
 $CopyCount = [int]$Database.Servers.Count
```

```
  if ($Database.MasterServerOrAvailabilityGroup.Name -ne $Database.Server.Name)
  {
   $Copies = [array]($Database.Servers | % { $_.Name })
  } else {
   $Copies = @()
  }
  # Archive Info
  $ArchiveMailboxCount = [int]([array]($ArchiveMailboxes | Where {$_.ArchiveDatabase -eq $Database.Name})).Count
  $ArchiveStatistics = [array]($ArchiveMailboxes | Where {$_.ArchiveDatabase -eq $Database.Name} | Get-MailboxStatistics -
Archive )
  if ($ArchiveStatistics)
  {
   [long]$ArchiveItemSizeB = 0
   $ArchiveStatistics | %{ $ArchiveItemSizeB+=$_.TotalItemSize.Value.ToBytes() }
   [long]$ArchiveAverageSize = $ArchiveItemSizeB / $ArchiveStatistics.Count
  } else {
   $ArchiveAverageSize = 0
  }
  # DB Size / Whitespace Info
  [long]$Size = $Database.DatabaseSize.ToBytes()
  [long]$Whitespace = $Database.AvailableNewMailboxSpace.ToBytes()
  $StorageGroup = $null


 } else {
  $ArchiveMailboxCount = 0
  $CopyCount = 0
  $Copies = @()
  # 2003 & 2007, Use WMI (Based on code by Gary Siepser, http://bit.ly/kWWMb3)
  $Size = [long](get-wmiobject cim_datafile -computername $Database.Server.Name -filter ('name=''' +
$Database.edbfilepath.pathname.replace("\","\\") + '''')).filesize

if (!$Size)
  {
   Write-Warning "Cannot detect database size via WMI for $($Database.Server.Name)"
   [long]$Size = 0
   [long]$Whitespace = 0
  } else {
   [long]$MailboxDeletedItemSizeB = 0
   if ($MailboxStatistics)
   {
    $MailboxStatistics | %{ $MailboxDeletedItemSizeB+=$_.TotalDeletedItemSizeB }
   }
   $Whitespace = $Size - $MailboxItemSizeB - $MailboxDeletedItemSizeB
   if ($Whitespace -lt 0) { $Whitespace = 0 }
  }
  $StorageGroup =$Database.DistinguishedName.Split(",")[1].Replace("CN=","")
 }

 @{Name      = $Database.Name
   StorageGroup    = $StorageGroup
   ActiveOwner    = $Database.Server.Name.ToUpper()
```

```
  MailboxCount    = [long]([array]($Mailboxes | Where {$_.Database -eq $Database.Identity})).Count
  MailboxAverageSize  = $MailboxAverageSize
  ArchiveMailboxCount  = $ArchiveMailboxCount
  ArchiveAverageSize  = $ArchiveAverageSize
  CircularLoggingEnabled  = $CircularLoggingEnabled
  LastFullBackup   = $LastFullBackup
  Size      = $Size
  Whitespace    = $Whitespace
  Copies     = $Copies
  CopyCount     = $CopyCount
  FreeLogDiskSpace   = $FreeLogDiskSpace
  FreeDatabaseDiskSpace  = $FreeDatabaseDiskSpace
  }
}


# Sub-Function to get mailbox count per server.
# New in 1.5.2
function _GetExSvrMailboxCount
{
 param($Mailboxes,$ExchangeServer,$Databases)
 # The following *should* work, but it doesn't. Apparently, ServerName is not always returned correctly which may be the
cause of
 # reports of counts being incorrect
 #([array]($Mailboxes | Where {$_.ServerName -eq $ExchangeServer.Name})).Count

 # ..So as a workaround, I'm going to check what databases are assigned to each server and then get the mailbox counts on a
per-
 # database basis and return the resulting total. As we already have this information resident in memory it should be cheap,
just
 # not as quick.
 $MailboxCount = 0
 foreach ($Database in [array]($Databases | Where {$_.Server -eq $ExchangeServer.Name}))
 {
  $MailboxCount+=([array]($Mailboxes | Where {$_.Database -eq $Database.Identity})).Count
 }
 $MailboxCount

}

# Sub-Function to Get Exchange Server information
function _GetExSvr
{
 param($E2010,$ExchangeServer,$Mailboxes,$Databases,$Hybrids)

 # Set Basic Variables
 $MailboxCount = 0
 $RollupLevel = 0
 $RollupVersion = ""
   $ExtNames = @()
   $IntNames = @()
   $CASArrayName = ""
```

```
 # Get WMI Information
 $tWMI = Get-WmiObject Win32_OperatingSystem -ComputerName $ExchangeServer.Name -ErrorAction SilentlyContinue
 if ($tWMI)
 {
  $OSVersion = $tWMI.Caption.Replace("(R)","").Replace("Microsoft
","").Replace("Enterprise","Ent").Replace("Standard","Std").Replace(" Edition","")
  $OSServicePack = $tWMI.CSDVersion
  $RealName = $tWMI.CSName.ToUpper()
 } else {
  Write-Warning "Cannot detect OS information via WMI for $($ExchangeServer.Name)"
  $OSVersion = "N/A"
  $OSServicePack = "N/A"
  $RealName = $ExchangeServer.Name.ToUpper()
 }
 $tWMI=Get-WmiObject -query "Select * from Win32_Volume" -ComputerName $ExchangeServer.Name -ErrorAction
SilentlyContinue
 if ($tWMI)
 {
  $Disks=$tWMI | Select Name,Capacity,FreeSpace | Sort-Object -Property Name
 } else {
  Write-Warning "Cannot detect OS information via WMI for $($ExchangeServer.Name)"
  $Disks=$null
 }

 # Get Exchange Version
 if ($ExchangeServer.AdminDisplayVersion.Major -eq 6)
 {
  $ExchangeMajorVersion =
"$($ExchangeServer.AdminDisplayVersion.Major).$($ExchangeServer.AdminDisplayVersion.Minor)"
  $ExchangeSPLevel = $ExchangeServer.AdminDisplayVersion.FilePatchLevelDescription.Replace("Service Pack ","")
 } elseif ($ExchangeServer.AdminDisplayVersion.Major -eq 15 -and $ExchangeServer.AdminDisplayVersion.Minor -eq 1) {
       $ExchangeMajorVersion =
[double]"$($ExchangeServer.AdminDisplayVersion.Major).$($ExchangeServer.AdminDisplayVersion.Minor)"
       $ExchangeSPLevel = 0
    } else {
  $ExchangeMajorVersion = $ExchangeServer.AdminDisplayVersion.Major
  $ExchangeSPLevel = $ExchangeServer.AdminDisplayVersion.Minor
 }
 # Exchange 2007+
 if ($ExchangeMajorVersion -ge 8)
 {
  # Get Roles
  $MailboxStatistics=$null
     [array]$Roles = $ExchangeServer.ServerRole.ToString().Replace(" ","").Split(",");
       # Add Hybrid "Role" for report
       if ($Hybrids -contains $ExchangeServer.Name)
       {
          $Roles+="Hybrid"
       }
  if ($Roles -contains "Mailbox")
  {
```

```
  $MailboxCount = _GetExSvrMailboxCount -Mailboxes $Mailboxes -ExchangeServer $ExchangeServer -Databases
$Databases
  if ($ExchangeServer.Name.ToUpper() -ne $RealName)
  {
   $Roles = [array]($Roles | Where {$_ -ne "Mailbox"})
   $Roles += "ClusteredMailbox"
  }
  # Get Mailbox Statistics the normal way, return in a consitent format
  $MailboxStatistics = Get-MailboxStatistics -Server $ExchangeServer | Select
DisplayName,@{Name="TotalItemSizeB";Expression=
{$_.TotalItemSize.Value.ToBytes()}},@{Name="TotalDeletedItemSizeB";Expression=
{$_.TotalDeletedItemSize.Value.ToBytes()}},Database
    }
      # Get HTTPS Names (Exchange 2010 only due to time taken to retrieve data)
      if ($Roles -contains "ClientAccess" -and $E2010)
      {

        Get-OWAVirtualDirectory -Server $ExchangeServer -ADPropertiesOnly | %{ $ExtNames+=$_.ExternalURL.Host;
$IntNames+=$_.InternalURL.Host; }
        Get-WebServicesVirtualDirectory -Server $ExchangeServer -ADPropertiesOnly | %{ $ExtNames+=$_.ExternalURL.Host;
$IntNames+=$_.InternalURL.Host; }
        Get-OABVirtualDirectory -Server $ExchangeServer -ADPropertiesOnly | %{ $ExtNames+=$_.ExternalURL.Host;
$IntNames+=$_.InternalURL.Host; }
        Get-ActiveSyncVirtualDirectory -Server $ExchangeServer -ADPropertiesOnly | %{ $ExtNames+=$_.ExternalURL.Host;
$IntNames+=$_.InternalURL.Host; }
        if (Get-Command Get-MAPIVirtualDirectory -ErrorAction SilentlyContinue)
        {
          Get-MAPIVirtualDirectory -Server $ExchangeServer -ADPropertiesOnly | %{ $ExtNames+=$_.ExternalURL.Host;
$IntNames+=$_.InternalURL.Host; }
        }
        if (Get-Command Get-ClientAccessService -ErrorAction SilentlyContinue)
        {
          $IntNames+=(Get-ClientAccessService -Identity $ExchangeServer.Name).AutoDiscoverServiceInternalURI.Host
        } else {
          $IntNames+=(Get-ClientAccessServer -Identity $ExchangeServer.Name).AutoDiscoverServiceInternalURI.Host
        }

        if ($ExchangeMajorVersion -ge 14)
        {
          Get-ECPVirtualDirectory -Server $ExchangeServer -ADPropertiesOnly | %{ $ExtNames+=$_.ExternalURL.Host;
$IntNames+=$_.InternalURL.Host; }
        }
        $IntNames = $IntNames|Sort-Object -Unique
        $ExtNames = $ExtNames|Sort-Object -Unique
        $CASArray = Get-ClientAccessArray -Site $ExchangeServer.Site.Name
        if ($CASArray)
        {
          $CASArrayName = $CASArray.Fqdn
        }
      }

  # Rollup Level / Versions (Thanks to Bhargav Shukla http://bit.ly/msxGIJ)
```

```
 if ($ExchangeMajorVersion -ge 14) {
        $RegKey="SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Installer\\UserData\\S-1-5-
18\\Products\\AE1D439464EB1B8488741FFA028E291C\\Patches"
      } else {
  $RegKey="SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Installer\\UserData\\S-1-5-
18\\Products\\461C2B4266EDEF444B864AD6D9E5B613\\Patches"
 }
 $RemoteRegistry = [Microsoft.Win32.RegistryKey]::OpenRemoteBaseKey('LocalMachine', $ExchangeServer.Name);
 if ($RemoteRegistry)
 {
  $RUKeys = $RemoteRegistry.OpenSubKey($RegKey).GetSubKeyNames() | ForEach {"$RegKey\\$_"}
  if ($RUKeys)
  {
   [array]($RUKeys | %{$RemoteRegistry.OpenSubKey($_).getvalue("DisplayName")}) | %{
    if ($_ -like "Update Rollup *")
    {
     $tRU = $_.Split(" ")[2]
     if ($tRU -like "*-*") { $tRUV=$tRU.Split("-")[1]; $tRU=$tRU.Split("-")[0] } else { $tRUV="" }
     if ([int]$tRU -ge [int]$RollupLevel) { $RollupLevel=$tRU; $RollupVersion=$tRUV }
    }
   }
  }
      } else {
  Write-Warning "Cannot detect Rollup Version via Remote Registry for $($ExchangeServer.Name)"
 }
      # Exchange 2013 CU or SP Level
      if ($ExchangeMajorVersion -ge 15)
 {
  $RegKey="SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Uninstall\\Microsoft Exchange v15"
    $RemoteRegistry = [Microsoft.Win32.RegistryKey]::OpenRemoteBaseKey('LocalMachine', $ExchangeServer.Name);
    if ($RemoteRegistry)
    {
     $ExchangeSPLevel = $RemoteRegistry.OpenSubKey($RegKey).getvalue("DisplayName")
           if ($ExchangeSPLevel -like "*Service Pack*" -or $ExchangeSPLevel -like "*Cumulative Update*")
           {
        $ExchangeSPLevel = $ExchangeSPLevel.Replace("Microsoft Exchange Server 2013 ","");
              $ExchangeSPLevel = $ExchangeSPLevel.Replace("Microsoft Exchange Server 2016 ","");
              $ExchangeSPLevel = $ExchangeSPLevel.Replace("Service Pack ","SP");
              $ExchangeSPLevel = $ExchangeSPLevel.Replace("Cumulative Update ","CU");
           } else {
              $ExchangeSPLevel = 0;
           }
        } else {
     Write-Warning "Cannot detect CU/SP via Remote Registry for $($ExchangeServer.Name)"
    }
      }

}
# Exchange 2003
if ($ExchangeMajorVersion -eq 6.5)
{
 # Mailbox Count
```

```
 $MailboxCount = _GetExSvrMailboxCount -Mailboxes $Mailboxes -ExchangeServer $ExchangeServer -Databases $Databases
 # Get Role via WMI
 $tWMI = Get-WMIObject Exchange_Server -Namespace "root\microsoftexchangev2" -Computername
$ExchangeServer.Name -Filter "Name='$($ExchangeServer.Name)'"
 if ($tWMI)
 {
  if ($tWMI.IsFrontEndServer) { $Roles=@("FE") } else { $Roles=@("BE") }
 } else {
  Write-Warning "Cannot detect Front End/Back End Server information via WMI for $($ExchangeServer.Name)"
  $Roles+="Unknown"
 }
 # Get Mailbox Statistics using WMI, return in a consistent format
 $tWMI = Get-WMIObject -class Exchange_Mailbox -Namespace ROOT\MicrosoftExchangev2 -ComputerName
$ExchangeServer.Name -Filter ("ServerName='$($ExchangeServer.Name)'")
 if ($tWMI)
 {
  $MailboxStatistics = $tWMI | Select @{Name="DisplayName";Expression=
{$_.MailboxDisplayName}},@{Name="TotalItemSizeB";Expression={$_.Size}},@{Name="TotalDeletedItemSizeB";Expression=
{$_.DeletedMessageSizeExtended }},@{Name="Database";Expression={((get-mailboxdatabase -Identity
"$($_.ServerName)\$($_.StorageGroupName)\$($_.StoreName)").identity)}}
 } else {
  Write-Warning "Cannot retrieve Mailbox Statistics via WMI for $($ExchangeServer.Name)"
  $MailboxStatistics = $null
 }
}
# Exchange 2000
if ($ExchangeMajorVersion -eq "6.0")
{
 # Mailbox Count
 $MailboxCount = _GetExSvrMailboxCount -Mailboxes $Mailboxes -ExchangeServer $ExchangeServer -Databases $Databases
 # Get Role via ADSI
 $tADSI=[ADSI]"LDAP://$($ExchangeServer.OriginatingServer)/$($ExchangeServer.DistinguishedName)"
 if ($tADSI)
 {
  if ($tADSI.ServerRole -eq 1) { $Roles=@("FE") } else { $Roles=@("BE") }
 } else {
  Write-Warning "Cannot detect Front End/Back End Server information via ADSI for $($ExchangeServer.Name)"
  $Roles+="Unknown"
 }
 $MailboxStatistics = $null
}

# Return Hashtable
@{Name     = $ExchangeServer.Name.ToUpper()
 RealName    = $RealName
 ExchangeMajorVersion  = $ExchangeMajorVersion
 ExchangeSPLevel  = $ExchangeSPLevel
 Edition    = $ExchangeServer.Edition
 Mailboxes    = $MailboxCount
 OSVersion    = $OSVersion;
 OSServicePack   = $OSServicePack
 Roles     = $Roles
```

```
  RollupLevel   = $RollupLevel
  RollupVersion   = $RollupVersion
  Site     = $ExchangeServer.Site.Name
  MailboxStatistics  = $MailboxStatistics
  Disks     = $Disks
     IntNames    = $IntNames
     ExtNames    = $ExtNames
     CASArrayName   = $CASArrayName
 }
}

# Sub Function to Get Totals by Version
function _TotalsByVersion
{
 param($ExchangeEnvironment)
 $TotalMailboxesByVersion=@{}
 if ($ExchangeEnvironment.Sites)
 {
  foreach ($Site in $ExchangeEnvironment.Sites.GetEnumerator())
  {
   foreach ($Server in $Site.Value)
   {
    if (!$TotalMailboxesByVersion["$($Server.ExchangeMajorVersion).$($Server.ExchangeSPLevel)"])
    {

$TotalMailboxesByVersion.Add("$($Server.ExchangeMajorVersion).$($Server.ExchangeSPLevel)",@{ServerCount=1;MailboxCo
unt=$Server.Mailboxes})
    } else {
     $TotalMailboxesByVersion["$($Server.ExchangeMajorVersion).$($Server.ExchangeSPLevel)"].ServerCount++

$TotalMailboxesByVersion["$($Server.ExchangeMajorVersion).$($Server.ExchangeSPLevel)"].MailboxCount+=$Server.Mailbox
es
    }
   }
  }
 }
 if ($ExchangeEnvironment.Pre2007)
 {
  foreach ($FakeSite in $ExchangeEnvironment.Pre2007.GetEnumerator())
  {
   foreach ($Server in $FakeSite.Value)
   {
    if (!$TotalMailboxesByVersion["$($Server.ExchangeMajorVersion).$($Server.ExchangeSPLevel)"])
    {

$TotalMailboxesByVersion.Add("$($Server.ExchangeMajorVersion).$($Server.ExchangeSPLevel)",@{ServerCount=1;MailboxCo
unt=$Server.Mailboxes})
    } else {
     $TotalMailboxesByVersion["$($Server.ExchangeMajorVersion).$($Server.ExchangeSPLevel)"].ServerCount++

$TotalMailboxesByVersion["$($Server.ExchangeMajorVersion).$($Server.ExchangeSPLevel)"].MailboxCount+=$Server.Mailbox
es
```

```
   }
  }
 }
}
 $TotalMailboxesByVersion
}

# Sub Function to Get Totals by Role
function _TotalsByRole
{
 param($ExchangeEnvironment)
 # Add Roles We Always Show
 $TotalServersByRole=@{"ClientAccess"   = 0
       "HubTransport"   = 0
       "UnifiedMessaging" = 0
       "Mailbox"    = 0
       "Edge"     = 0
       }
 if ($ExchangeEnvironment.Sites)
 {
  foreach ($Site in $ExchangeEnvironment.Sites.GetEnumerator())
  {
   foreach ($Server in $Site.Value)
   {
    foreach ($Role in $Server.Roles)
    {
     if ($TotalServersByRole[$Role] -eq $null)
     {
      $TotalServersByRole.Add($Role,1)
     } else {
      $TotalServersByRole[$Role]++
     }
    }
   }
  }
 }
 if ($ExchangeEnvironment.Pre2007["Pre 2007 Servers"])
 {

  foreach ($Server in $ExchangeEnvironment.Pre2007["Pre 2007 Servers"])
  {

   foreach ($Role in $Server.Roles)
   {
    if ($TotalServersByRole[$Role] -eq $null)
    {
     $TotalServersByRole.Add($Role,1)
    } else {
     $TotalServersByRole[$Role]++
    }
   }
  }
```

```
 }
 $TotalServersByRole
}

# Sub Function to return HTML Table for Sites/Pre 2007
function _GetOverview
{
 param($Servers,$ExchangeEnvironment,$ExRoleStrings,$Pre2007=$False)
 if ($Pre2007)
 {
  $BGColHeader="#880099"
  $BGColSubHeader="#8800CC"
  $Prefix=""
      $IntNamesText=""
      $ExtNamesText=""
      $CASArrayText=""
 } else {
  $BGColHeader="#000099"
  $BGColSubHeader="#0000FF"
  $Prefix="Site:"
      $IntNamesText=""
      $ExtNamesText=""
      $CASArrayText=""
      $IntNames=@()
      $ExtNames=@()
      $CASArrayName=""
      foreach ($Server in $Servers.Value)
      {
         $IntNames+=$Server.IntNames
         $ExtNames+=$Server.ExtNames
         $CASArrayName=$Server.CASArrayName

      }
      $IntNames = $IntNames|Sort -Unique
      $ExtNames = $ExtNames|Sort -Unique
      $IntNames = [system.String]::Join(",",$IntNames)
      $ExtNames = [system.String]::Join(",",$ExtNames)
      if ($IntNames)
      {
         $IntNamesText="Internal Names: $($IntNames)"
         $ExtNamesText="External Names: $($ExtNames)<br >"
      }
      if ($CASArrayName)
      {
         $CASArrayText="CAS Array: $($CASArrayName)"
      }
 }
 $Output="<table border=""0"" cellpadding=""3"" width=""100%"" style=""font-size:8pt;font-family:Segoe UI,Arial,sans-
serif"">
 <col width=""20%""><col width=""20%"">
 <colgroup width=""25%"">";
```

```
 $ExchangeEnvironment.TotalServersByRole.GetEnumerator()|Sort Name| %{$Output+="<col width=""3%"">"}
 $Output+="</colgroup><col width=""20%""><col  width=""20%"">
 <tr bgcolor=""$($BGColHeader)""><th><font color=""#ffffff"">$($Prefix) $($Servers.Key)</font></th>
 <th colspan=""$(($ExchangeEnvironment.TotalServersByRole.Count)+2)"" align=""left""><font
color=""#ffffff"">$($ExtNamesText)$($IntNamesText)</font></th>
 <th align=""center""><font color=""#ffffff"">$($CASArrayText)</font></th></tr>"
 $TotalMailboxes=0
 $Servers.Value | %{$TotalMailboxes += $_.Mailboxes}
 $Output+="<tr bgcolor=""$($BGColSubHeader)""><th><font color=""#ffffff"">Mailboxes: $($TotalMailboxes)</font></th>
<th>"
    $Output+="<font color=""#ffffff"">Exchange Version</font></th>"
 $ExchangeEnvironment.TotalServersByRole.GetEnumerator()|Sort Name| %{$Output+="<th><font
color=""#ffffff"">$($ExRoleStrings[$_.Key].Short)</font></th>"}
 $Output+="<th><font color=""#ffffff"">OS Version</font></th><th><font color=""#ffffff"">OS Service Pack</font></th>
</tr>"
 $AlternateRow=0

 foreach ($Server in $Servers.Value)
 {
  $Output+="<tr "
  if ($AlternateRow)
  {
   $Output+=" style=""background-color:#dddddd"""
   $AlternateRow=0
  } else
  {
   $AlternateRow=1
  }
  $Output+="><td>$($Server.Name)"
  if ($Server.RealName -ne $Server.Name)
  {
   $Output+=" ($($Server.RealName))"
  }
  $Output+="</td><td>$($ExVersionStrings["$($Server.ExchangeMajorVersion).$($Server.ExchangeSPLevel)"].Long)"
  if ($Server.RollupLevel -gt 0)
  {
   $Output+=" UR$($Server.RollupLevel)"
   if ($Server.RollupVersion)
   {
    $Output+=" $($Server.RollupVersion)"
   }
  }
  $Output+="</td>"
  $ExchangeEnvironment.TotalServersByRole.GetEnumerator()|Sort Name| %{
   $Output+="<td"
   if ($Server.Roles -contains $_.Key)
   {
    $Output+=" align=""center"" style=""background-color:#00FF00"""
   }
   $Output+=">"
   if (($_.Key -eq "ClusteredMailbox" -or $_.Key -eq "Mailbox" -or $_.Key -eq "BE") -and $Server.Roles -contains $_.Key)
   {
```

```
    $Output+=$Server.Mailboxes
   }
  }

  $Output+="<td>$($Server.OSVersion)</td><td>$($Server.OSServicePack)</td></tr>";
 }
 $Output+="<tr></tr>
 </table><br />"
 $Output
}

# Sub Function to return HTML Table for Databases
function _GetDBTable
{

 param($Databases)
 # Only Show Archive Mailbox Columns, Backup Columns and Circ Logging if at least one DB has an Archive mailbox, backed
up or Cir Log enabled.
 $ShowArchiveDBs=$False
 $ShowLastFullBackup=$False
 $ShowCircularLogging=$False
 $ShowStorageGroups=$False
 $ShowCopies=$False
 $ShowFreeDatabaseSpace=$False
 $ShowFreeLogDiskSpace=$False
 foreach ($Database in $Databases)
 {
  if ($Database.ArchiveMailboxCount -gt 0)
  {
   $ShowArchiveDBs=$True
  }
  if ($Database.LastFullBackup -ne "Not Available")
  {
   $ShowLastFullBackup=$True
  }
  if ($Database.CircularLoggingEnabled -eq "Yes")
  {
   $ShowCircularLogging=$True
  }
  if ($Database.StorageGroup)
  {
   $ShowStorageGroups=$True
  }
  if ($Database.CopyCount -gt 0)
  {
   $ShowCopies=$True
  }
  if ($Database.FreeDatabaseDiskSpace -ne $null)
  {
   $ShowFreeDatabaseSpace=$true
  }
  if ($Database.FreeLogDiskSpace -ne $null)
```

```
 {
  $ShowFreeLogDiskSpace=$true
 }

}


$Output="<table border=""0"" cellpadding=""3"" width=""100%"" style=""font-size:8pt;font-family:Segoe UI,Arial,sans-serif"">

<tr align=""center"" bgcolor=""#FFD700"">

<th>Server</th>"

if ($ShowStorageGroups)
{
 $Output+="<th>Storage Group</th>"
}
$Output+="<th>Database Name</th>
<th>Mailboxes</th>
<th>Av. Mailbox Size</th>"
if ($ShowArchiveDBs)
{
 $Output+="<th>Archive MBs</th><th>Av. Archive Size</th>"
}
$Output+="<th>DB Size</th><th>DB Whitespace</th>"
if ($ShowFreeDatabaseSpace)
{
 $Output+="<th>Database Disk Free</th>"
}
if ($ShowFreeLogDiskSpace)
{
 $Output+="<th>Log Disk Free</th>"
}
if ($ShowLastFullBackup)
{
 $Output+="<th>Last Full Backup</th>"
}
if ($ShowCircularLogging)
{
 $Output+="<th>Circular Logging</th>"
}
if ($ShowCopies)
{
 $Output+="<th>Copies (n)</th>"
}

$Output+="</tr>"
$AlternateRow=0;
foreach ($Database in $Databases)
{
 $Output+="<tr"
```

```
if ($AlternateRow)
{
 $Output+=" style=""background-color:#dddddd"""
 $AlternateRow=0

        if($Database.Size/1GB -gt 2000)

        {
        $Output+=" style=""background-color:#FF0000"""
 $AlternateRow=0
        }
} else
{
 $AlternateRow=1
}

$Output+="><td>$($Database.ActiveOwner)</td>"
if ($ShowStorageGroups)
{
 $Output+="<td>$($Database.StorageGroup)</td>"
}
$Output+="<td>$($Database.Name)</td>
<td align=""center"">$($Database.MailboxCount)</td>
<td align=""center"">$("{0:N2}" -f ($Database.MailboxAverageSize/1MB)) MB</td>"
if ($ShowArchiveDBs)
{
 $Output+="<td align=""center"">$($Database.ArchiveMailboxCount)</td>
 <td align=""center"">$("{0:N2}" -f ($Database.ArchiveAverageSize/1MB)) MB</td>";
}

$Output+="<td align=""center"">$("{0:N2}" -f ($Database.Size/1GB)) GB </td>
<td align=""center"">$("{0:N2}" -f ($Database.Whitespace/1GB)) GB</td>";

if ($ShowFreeDatabaseSpace)
{
 $Output+="<td align=""center"">$("{0:N1}" -f $Database.FreeDatabaseDiskSpace)%</td>"
}
if ($ShowFreeLogDiskSpace)
{
 $Output+="<td align=""center"">$("{0:N1}" -f $Database.FreeLogDiskSpace)%</td>"
}
if ($ShowLastFullBackup)
{
 $Output+="<td align=""center"">$($Database.LastFullBackup)</td>";
}
if ($ShowCircularLogging)
{
 $Output+="<td align=""center"">$($Database.CircularLoggingEnabled)</td>";
}
if ($ShowCopies)
{
 $Output+="<td>$($Database.Copies|%{$_}) ($($Database.CopyCount))</td>"
```

```
  }
  $Output+="</tr>";
 }
 $Output+="</table><br />"

 $Output
}


# Sub Function to neatly update progress
function _UpProg1
{
 param($PercentComplete,$Status,$Stage)
 $TotalStages=5
 Write-Progress -id 1 -activity "Get-ExchangeEnvironmentReport" -status $Status -percentComplete
(($PercentComplete/$TotalStages)+(1/$TotalStages*$Stage*100))
}

# 1. Initial Startup

# 1.0 Check Powershell Version
if ((Get-Host).Version.Major -eq 1)
{
 throw "Powershell Version 1 not supported";
}

# 1.1 Check Exchange Management Shell, attempt to load
if (!(Get-Command Get-ExchangeServer -ErrorAction SilentlyContinue))
{
 if (Test-Path "C:\Program Files\Microsoft\Exchange Server\V14\bin\RemoteExchange.ps1")
 {
  . 'C:\Program Files\Microsoft\Exchange Server\V14\bin\RemoteExchange.ps1'
  Connect-ExchangeServer -auto
 } elseif (Test-Path "C:\Program Files\Microsoft\Exchange Server\bin\Exchange.ps1") {
  Add-PSSnapIn Microsoft.Exchange.Management.PowerShell.Admin
  .'C:\Program Files\Microsoft\Exchange Server\bin\Exchange.ps1'
 } else {
  throw "Exchange Management Shell cannot be loaded"
 }
}

# 1.2 Check if -SendMail parameter set and if so check -MailFrom, -MailTo and -MailServer are set
if ($SendMail)
{
 if (!$MailFrom -or !$MailTo -or !$MailServer)
 {
  throw "If -SendMail specified, you must also specify -MailFrom, -MailTo and -MailServer"
 }
}

# 1.3 Check Exchange Management Shell Version
if ((Get-PSSnapin -Name Microsoft.Exchange.Management.PowerShell.Admin -ErrorAction SilentlyContinue))
```

```
{
 $E2010 = $false;
 if (Get-ExchangeServer | Where {$_.AdminDisplayVersion.Major -gt 14})
 {
  Write-Warning "Exchange 2010 or higher detected. You'll get better results if you run this script from the latest management
shell"
 }
}else{

   $E2010 = $true
   $localserver = get-exchangeserver $Env:computername
   $localversion = $localserver.admindisplayversion.major
   if ($localversion -eq 15) { $E2013 = $true }

}

# 1.4 Check view entire forest if set (by default, true)
if ($E2010)
{
 Set-ADServerSettings -ViewEntireForest:$ViewEntireForest
} else {
 $global:AdminSessionADSettings.ViewEntireForest = $ViewEntireForest
}

# 1.5 Initial Variables

# 1.5.1 Hashtable to update with environment data
$ExchangeEnvironment = @{Sites     = @{}
      Pre2007    = @{}
      Servers    = @{}
      DAGs     = @()
      NonDAGDatabases  = @()
     }
# 1.5.7 Exchange Major Version String Mapping
$ExMajorVersionStrings = @{"6.0" = @{Long="Exchange 2000";Short="E2000"}
          "6.5" = @{Long="Exchange 2003";Short="E2003"}
          "8"   = @{Long="Exchange 2007";Short="E2007"}
                    "14"  = @{Long="Exchange 2010";Short="E2010"}
        "15"  = @{Long="Exchange 2013";Short="E2013"}
                    "15.1"  = @{Long="Exchange 2016";Short="E2016"}}
# 1.5.8 Exchange Service Pack String Mapping
$ExSPLevelStrings = @{"0" = "RTM"
      "1" = "SP1"
        "2" = "SP2"
          "3" = "SP3"
        "4" = "SP4"
                "SP1" = "SP1"
                "SP2" = "SP2"}
   # Add many CUs
   for ($i = 1; $i -le 20; $i++)
   {
      $ExSPLevelStrings.Add("CU$($i)","CU$($i)");
```

```
   }
# 1.5.9 Populate Full Mapping using above info
$ExVersionStrings = @{}
foreach ($Major in $ExMajorVersionStrings.GetEnumerator())
{
 foreach ($Minor in $ExSPLevelStrings.GetEnumerator())
 {
  $ExVersionStrings.Add("$($Major.Key).$($Minor.Key)",@{Long="$($Major.Value.Long)
$($Minor.Value)";Short="$($Major.Value.Short)$($Minor.Value)"})
 }
}
# 1.5.10 Exchange Role String Mapping
$ExRoleStrings = @{"ClusteredMailbox" = @{Short="ClusMBX";Long="CCR/SCC Clustered Mailbox"}
      "Mailbox"    = @{Short="MBX";Long="Mailbox"}
      "ClientAccess"   = @{Short="CAS";Long="Client Access"}
      "HubTransport"   = @{Short="HUB";Long="Hub Transport"}
      "UnifiedMessaging" = @{Short="UM";Long="Unified Messaging"}
      "Edge"     = @{Short="EDGE";Long="Edge Transport"}
      "FE"     = @{Short="FE";Long="Front End"}
      "BE"     = @{Short="BE";Long="Back End"}
            "Hybrid"       = @{Short="HYB"; Long="Hybrid"}
      "Unknown"   = @{Short="Unknown";Long="Unknown"}}

# 2 Get Relevant Exchange Information Up-Front

# 2.1 Get Server, Exchange and Mailbox Information
_UpProg1 1 "Getting Exchange Server List" 1
$ExchangeServers = [array](Get-ExchangeServer $ServerFilter)
if (!$ExchangeServers)
{
 throw "No Exchange Servers matched by -ServerFilter ""$($ServerFilter)"""
}
$HybridServers=@()
if (Get-Command Get-HybridConfiguration -ErrorAction SilentlyContinue)
{
   $HybridConfig = Get-HybridConfiguration
   $HybridConfig.ReceivingTransportServers|%{ $HybridServers+=$_.Name  }
   $HybridConfig.SendingTransportServers|%{ $HybridServers+=$_.Name  }
   $HybridServers =  $HybridServers | Sort-Object -Unique
}

_UpProg1 10 "Getting Mailboxes" 1
$Mailboxes = [array](Get-Mailbox -ResultSize Unlimited) | Where {$_.Server -like $ServerFilter}
if ($E2010)
{
 _UpProg1 60 "Getting Archive Mailboxes" 1
 $ArchiveMailboxes = [array](Get-Mailbox -Archive -ResultSize Unlimited) | Where {$_.Server -like $ServerFilter}
   _UpProg1 70 "Getting Remote Mailboxes" 1
   $RemoteMailboxes = [array](Get-RemoteMailbox  -ResultSize Unlimited)
   $ExchangeEnvironment.Add("RemoteMailboxes",$RemoteMailboxes.Count)
 _UpProg1 90 "Getting Databases" 1
   if ($E2013)
```

```
   {
      $Databases = [array](Get-MailboxDatabase -IncludePreExchange2013 -Status)  | Where {$_.Server -like $ServerFilter}
   }
   elseif ($E2010)
   {
      $Databases = [array](Get-MailboxDatabase -IncludePreExchange2010 -Status)  | Where {$_.Server -like $ServerFilter}
   }
 $DAGs = [array](Get-DatabaseAvailabilityGroup) | Where {$_.Servers -like $ServerFilter}
} else {
 $ArchiveMailboxes = $null
 $ArchiveMailboxStats = $null
 $DAGs = $null
 _UpProg1 90 "Getting Databases" 1
 $Databases = [array](Get-MailboxDatabase -IncludePreExchange2007 -Status) | Where {$_.Server -like $ServerFilter}
   $ExchangeEnvironment.Add("RemoteMailboxes",0)
}

# 2.3 Populate Information we know
$ExchangeEnvironment.Add("TotalMailboxes",$Mailboxes.Count + $ExchangeEnvironment.RemoteMailboxes);

# 3 Process High-Level Exchange Information

# 3.1 Collect Exchange Server Information
for ($i=0; $i -lt $ExchangeServers.Count; $i++)
{
 _UpProg1 ($i/$ExchangeServers.Count*100) "Getting Exchange Server Information" 2
 # Get Exchange Info
 $ExSvr = _GetExSvr -E2010 $E2010 -ExchangeServer $ExchangeServers[$i] -Mailboxes $Mailboxes -Databases $Databases -
Hybrids $HybridServers
 # Add to site or pre-Exchange 2007 list
 if ($ExSvr.Site)
 {
  # Exchange 2007 or higher
  if (!$ExchangeEnvironment.Sites[$ExSvr.Site])
  {
   $ExchangeEnvironment.Sites.Add($ExSvr.Site,@($ExSvr))
  } else {
   $ExchangeEnvironment.Sites[$ExSvr.Site]+=$ExSvr
  }
 } else {
  # Exchange 2003 or lower
  if (!$ExchangeEnvironment.Pre2007["Pre 2007 Servers"])
  {
   $ExchangeEnvironment.Pre2007.Add("Pre 2007 Servers",@($ExSvr))
  } else {
   $ExchangeEnvironment.Pre2007["Pre 2007 Servers"]+=$ExSvr
  }
 }
 # Add to Servers List
 $ExchangeEnvironment.Servers.Add($ExSvr.Name,$ExSvr)
}
```

```
# 3.2 Calculate Environment Totals for Version/Role using collected data
_UpProg1 1 "Getting Totals" 3
$ExchangeEnvironment.Add("TotalMailboxesByVersion",(_TotalsByVersion -ExchangeEnvironment $ExchangeEnvironment))
$ExchangeEnvironment.Add("TotalServersByRole",(_TotalsByRole -ExchangeEnvironment $ExchangeEnvironment))

# 3.4 Populate Environment DAGs
_UpProg1 5 "Getting DAG Info" 3
if ($DAGs)
{
 foreach($DAG in $DAGs)
 {
  $ExchangeEnvironment.DAGs+=(_GetDAG -DAG $DAG)
 }
}

# 3.5 Get Database information
_UpProg1 60 "Getting Database Info" 3
for ($i=0; $i -lt $Databases.Count; $i++)
{
 $Database = _GetDB -Database $Databases[$i] -ExchangeEnvironment $ExchangeEnvironment -Mailboxes $Mailboxes -
ArchiveMailboxes $ArchiveMailboxes -E2010 $E2010
 $DAGDB = $false
 for ($j=0; $j -lt $ExchangeEnvironment.DAGs.Count; $j++)
 {
  if ($ExchangeEnvironment.DAGs[$j].Members -contains $Database.ActiveOwner)
  {
   $DAGDB=$true
   $ExchangeEnvironment.DAGs[$j].Databases += $Database
  }
 }
 if (!$DAGDB)
 {
  $ExchangeEnvironment.NonDAGDatabases += $Database
 }


}

# 4 Write Information
_UpProg1 5 "Writing HTML Report Header" 4
# Header
$Output="<html>
<body>
<font size=""1"" face=""Segoe UI,Arial,sans-serif"">
<h2 align=""center"">[每月巡检]邮件系统环境检查报告</h3>
<h3 align=""center"" bgcolor=""#FF0000"">阅读须知：请重点关注被标记为深红色的行，深红色的行代表数据库占用空间已超过
2TB，需要优先做数据迁移。</h4>
<h4 align=""center"">生成时间 $((Get-Date).ToString())</h5>
</font>
<table border=""0"" cellpadding=""3"" style=""font-size:8pt;font-family:Segoe UI,Arial,sans-serif"">
<tr bgcolor=""#009900"">
<th colspan=""$($ExchangeEnvironment.TotalMailboxesByVersion.Count)""><font color=""#ffffff"">Total Servers:</font>
```

```
</th>"
if ($ExchangeEnvironment.RemoteMailboxes)
   {
   $Output+="<th colspan=""$($ExchangeEnvironment.TotalMailboxesByVersion.Count+2)""><font color=""#ffffff"">Total
Mailboxes:</font></th>"
   } else {
   $Output+="<th colspan=""$($ExchangeEnvironment.TotalMailboxesByVersion.Count+1)""><font color=""#ffffff"">Total
Mailboxes:</font></th>"
   }
$Output+="<th colspan=""$($ExchangeEnvironment.TotalServersByRole.Count)""><font color=""#ffffff"">Total Roles:
</font></th></tr>
<tr bgcolor=""#00CC00"">"
# Show Column Headings based on the Exchange versions we have
$ExchangeEnvironment.TotalMailboxesByVersion.GetEnumerator()|Sort Name| %{$Output+="
<th>$($ExVersionStrings[$_.Key].Short)</th>"}
$ExchangeEnvironment.TotalMailboxesByVersion.GetEnumerator()|Sort Name| %{$Output+="
<th>$($ExVersionStrings[$_.Key].Short)</th>"}
if ($ExchangeEnvironment.RemoteMailboxes)
{
   $Output+="<th>Office 365</th>"
}
$Output+="<th>Org</th>"
$ExchangeEnvironment.TotalServersByRole.GetEnumerator()|Sort Name| %{$Output+="<th>$($ExRoleStrings[$_.Key].Short)
</th>"}
$Output+="<tr>"
$Output+="<tr align=""center"" bgcolor=""#dddddd"">"
$ExchangeEnvironment.TotalMailboxesByVersion.GetEnumerator()|Sort Name| %{$Output+="<td>$($_.Value.ServerCount)
</td>" }
$ExchangeEnvironment.TotalMailboxesByVersion.GetEnumerator()|Sort Name| %{$Output+="<td>$($_.Value.MailboxCount)
</td>" }
if ($RemoteMailboxes)
{
   $Output+="<th>$($ExchangeEnvironment.RemoteMailboxes)</th>"
}
$Output+="<td>$($ExchangeEnvironment.TotalMailboxes)</td>"
$ExchangeEnvironment.TotalServersByRole.GetEnumerator()|Sort Name| %{$Output+="<td>$($_.Value)</td>"}
$Output+="</tr><tr><tr></table><br>"

# Sites and Servers
_UpProg1 20 "Writing HTML Site Information" 4
foreach ($Site in $ExchangeEnvironment.Sites.GetEnumerator())
{
 $Output+=_GetOverview -Servers $Site -ExchangeEnvironment $ExchangeEnvironment -ExRoleStrings $ExRoleStrings
}
_UpProg1 40 "Writing HTML Pre-2007 Information" 4
foreach ($FakeSite in $ExchangeEnvironment.Pre2007.GetEnumerator())
{
 $Output+=_GetOverview -Servers $FakeSite -ExchangeEnvironment $ExchangeEnvironment -ExRoleStrings $ExRoleStrings -
Pre2007:$true
}

_UpProg1 60 "Writing HTML DAG Information" 4
```

```
foreach ($DAG in $ExchangeEnvironment.DAGs)
{
 if ($DAG.MemberCount -gt 0)
 {
  # Database Availability Group Header
  $Output+="<table border=""0"" cellpadding=""3"" width=""100%"" style=""font-size:8pt;font-family:Segoe UI,Arial,sans-serif"">
  <col width=""20%""><col width=""10%""><col width=""70%"">
  <tr align=""center"" bgcolor=""#FF8000 ""><th>Database Availability Group Name</th><th>Member Count</th>
  <th>Database Availability Group Members</th></tr>
  <tr><td>$($DAG.Name)</td><td align=""center"">
  $($DAG.MemberCount)</td><td>"
  $DAG.Members | %{ $Output+="$($_) " }
  $Output+="</td></tr></table>"

  # Get Table HTML
  $Output+=_GetDBTable -Databases $DAG.Databases
 }

}

if ($ExchangeEnvironment.NonDAGDatabases.Count)
{
 _UpProg1 80 "Writing HTML Non-DAG Database Information" 4
 $Output+="<table border=""0"" cellpadding=""3"" width=""100%"" style=""font-size:8pt;font-family:Segoe UI,Arial,sans-serif"">
      <tr bgcolor=""#FF8000""><th>Mailbox Databases (Non-DAG)</th></table>"
 $Output+=_GetDBTable -Databases $ExchangeEnvironment.NonDAGDatabases
}


# End
_UpProg1 90 "Finishing off.." 4
$Output+="</body></html>";
$Output | Out-File $HTMLReport


<#if ($SendMail)
{
 _UpProg1 95 "Sending mail message.." 4
 Send-MailMessage -Attachments $HTMLReport -To $MailTo -From $MailFrom -Subject "[每月巡检]邮件系统环境检查报告" -BodyAsHtml $Output -SmtpServer $MailServer -Encoding default
}#>

$users="administrator@demo.com"
if ($HTMLReport -ne "")
{
   foreach ($user in $users)
{
      Write-Host "Sending Email notification to $user"

  $smtpServer = "172.168.1.50"
```

```
$smtp = New-Object Net.Mail.SmtpClient($smtpServer)
$msg = New-Object Net.Mail.MailMessage
$msg.To.Add($user)
    $msg.From = "zengchuixin101@demo.com"
$msg.Subject = "[系统组每月巡检]邮件系统环境检查报告"
    $msg.IsBodyHTML = $true
    $msg.Body = get-content $HTMLReport
$smtp.Send($msg)
    $body = "邮件系统环境检查报告"
 }
}
```