

```
<#
.DESCRIPTION
    This module contains the common functions that are used by many HSC PowerShell files
    These functions are:

    1. Set-Environment
    2. Set-WindowTitle
    3. Get-Parameter
    4. Start-Countdown
    5. Test-Verbose
    6. Write-ColorOutput
    7. Get-LogFileName
    8. Test-LogFilePath
    9. Remove-OldLogFile
    10. Write-LogFileSummaryInformation
    11. Send-Email
    12. Get-PasswordFromSecureStringFile
    13. Get-RandomPassword
    14. Exit-Command
    15. Test-ValidWVUEmail

.NOTES
    HSC-CommonCodeModule.psm1
    Last Modified by: Jeff Brusoe
    Last Modified: June 8, 2020

    Version: 1.5
#>

[CmdletBinding()]
[.Diagnostics.CodeAnalysis.SuppressMessageAttribute("PSAvoidTrailingWhiteSpace","",Justification = "Not relevant")]
param ()

function Set-Environment
{
    <#
    .DESCRIPTION
        This function configures the environment for files to use this module. It performs
        the following tasks.
        1. Sets strictmode to the latest version
        2. Clear $Error variable
        3. Clear PS window
        4. Sets the PowerShell window title
        5. Set location to root of ps1 directory
        6. Generates transcript log file path
        7. Start transcript log file
        8. Removes old .txt log files
        9. Set $ErrorActionPreference

    .NOTES
        Written by: Jeff Brusoe
        Last Updated by: Jeff Brusoe
        Last Updated: June 2, 2020
```

```
52 | #>
53 |
54 | [CmdletBinding()]
55 |
56 | [Diagnostics.CodeAnalysis.SuppressMessageAttribute("PSUseShouldProcessForStateChanging
unctions", "", Justification = "Doesn't make serious state changes")]
57 | param (
58 |     [bool]$NoSessionTranscript=$false,
59 |     [string]$LogFilePath = $($MyInvocation.PSScriptRoot + "\Logs\"),
60 |     [bool]$StopOnError,
61 |     [int]$DaysToKeepLogFiles = 5
62 | )
63 |
64 | #1. Set strict mode
65 | Set-StrictMode -Version Latest #Configures for current scope (Probably not needed)
66 | Set-PSDebug -Strict #Configures struct mode for global scope
67 |
68 | #2. Clear Error Variable
69 | $global:Error.Clear()
70 |
71 | #3. Clear PS Window
72 | Clear-Host
73 |
74 | #4. Set Window Title
75 | Set-WindowTitle -WindowTitle $MyInvocation.PSCommandPath
76 |
77 | #5. Set location to $PSScriptRoot
78 | Set-Location $MyInvocation.PSScriptRoot #Don't use $PSScriptRoot here since that put
it in the common code directory instead of the script root directory.
79 |
80 | #6. Generate transcript log file path
81 | #Parse file location to determine program name
82 | Write-Output $("PSCommandPath: " + $MyInvocation.PSCommandPath) | Out-Host
83 | $ProgramName = $MyInvocation.PSCommandPath
84 | $ProgramName = $ProgramName.substring(0,$ProgramName.indexOf("."))
85 | $ProgramName = $ProgramName.substring($ProgramName.lastIndexOf("\")+1)
86 | Write-Output "Program Name: $ProgramName" | Out-Host
87 |
88 | $TranscriptLogFile = Get-LogFileName -ProgramName $ProgramName
89 | Write-Output "Transcript File Path: $TranscriptLogFile" | Out-Host
90 |
91 | #7 & 8. Start transcript and remove old log files
92 | if (Test-LogFilePath -LogFilePath $LogFilePath)
93 | {
94 |     if (!$NoSessionTranscript)
95 |     {
96 |         Write-Verbose "Starting transcript log file" | Out-Host
97 |         Start-Transcript $TranscriptLogFile | Out-Host
98 |     }
99 |     else
100 |     {
101 |         Write-Output "Transcript log file will not be created..." | Out-Host
102 |     }
103 |
104 |     Write-Output "Removing old log files" | Out-Host
```

```
104     Remove-OldLogFile -CSV -TXT -Path $LogFilePath -Days $DaysToKeepLogFiles -Verbose
-Delete
105 }
106 else
107 {
108     Write-Output "Log file path doesn't exist..." | Out-Host
109 }
110
111 #9. Set $ErrorActionPreference
112 if ($StopOnError)
113 {
114     $global:ErrorActionPreference = "Stop"
115 }
116 else
117 {
118     $global:ErrorActionPreference = "Continue"
119 }
120 }
121
122 function Set-WindowTitle
123 {
124     <#
125     .DESCRIPTION
126     The purpose of this function is to change the title in the PowerShell window.
127     It can do this by either passing in a value or by parsing up the file path.
128
129     .PARAMETER WindowTitle
130     This is a string parameter that specifies the PowerShell window title. If it
131     isn't provided, it will be determined by the $PSCommandPath variable.
132
133     .NOTES
134     Written by: Jeff Brusoe
135     Last Updated by: Jeff Brusoe
136     Last Updated: June 2, 2020
137     #>
138
139     [CmdletBinding()]
140
141     [Diagnostics.CodeAnalysis.SuppressMessageAttribute("PSUseShouldProcessForStateChanging
unctions", "", Justification = "Start-Sleep Doesn't Change System State.")]
142     param (
143         [string]$WindowTitle=$MyInvocation.PSCommandPath #Full path with file name
144     )
145
146     if (![string]::IsNullOrEmpty($WindowTitle))
147     {
148         try
149         {
150             Write-Verbose "Setting window title" | Out-Host
151             $WindowTitle = $WindowTitle.substring($WindowTitle.lastIndexOf("\")+1)
152             Write-Verbose $WindowTitle | Out-Host
153
154             $Host.UI.RawUI.WindowTitle = $WindowTitle
155         }
156         catch
157         {
158         }
159     }
160 }
```

```
156     {
157         Write-Warning "Unable to set the window title" | Out-Host
158     }
159 }
160 }
161
162 function Get-Parameter
163 {
164     <#
165     .DESCRIPTION
166         The purpose of this function is to display any nondefault
167         parameters that were passed to the originating function.
168
169     .PARAMETER ParameterList
170         This parameter comes from the built-in $PSBoundParameters variable.
171         See: https://blogs.msdn.microsoft.com/timid/2014/08/12/psboundparameters-and-
172         commonparameters-whatif-debug-etc/
173
174     .NOTES
175         Written by: Jeff Brusoe
176         Last Updated by: Jeff Brusoe
177         Last Updated: June 3, 2020
178     #>
179     [CmdletBinding()]
180     param (
181         [Parameter(Mandatory=$true)][hashtable]$ParameterList
182     )
183
184     try
185     {
186         if (($ParameterList.keys | Measure-Object).Count -eq 0)
187         {
188             Write-Output "All input parameters are set to default values." | Out-Host
189         }
190         else
191         {
192             Write-Output "The following parameters have nondefault values:" | Out-Host
193
194             foreach ($key in $ParameterList.keys)
195             {
196                 $param = Get-Variable -Name $key -ErrorAction SilentlyContinue
197
198                 if($null -ne $param)
199                 {
200                     Write-Output "$($param.name): $($param.value)" | Out-Host
201                 }
202             }
203         }
204     }
205     catch
206     {
207         Write-Warning "There was an error generating the parameter list." | Out-Host
208     }
209 }
```

```
210 Write-Output "`n" | Out-Host
211 }
212
213 Function Start-Countdown
214 {
215     <#
216     .DESCRIPTION
217         This function displays a progress bar and message stating the reason for the delay
218         It is basically a more user friendly version of Start-Sleep which may look like th
window
219         has locked up if it is used.
220
221     .PARAMETER Seconds
222         This is the integer value that tells how long the pause should occur for.
223
224     .PARAMETER Message
225
226     .NOTES
227         Written by: Jeff Brusoe
228         Last Updated by: Jeff Brusoe
229         Last Updated: October 21, 2016
230     #>
231
232     [Diagnostics.CodeAnalysis.SuppressMessageAttribute("PSUseShouldProcessForStateChanging
unctions", "", Justification = "Start-Sleep Doesn't Change System State.")]
233     [CmdletBinding(PositionalBinding=$false)]
234
235     Param(
236         [Parameter(ValueFromPipeline=$true)][Int32]$Seconds = 10,
237         [Parameter(ValueFromPipeline=$true)][string]$Message = "Pausing for $Seconds
seconds..."
238     )
239
240     process
241     {
242         for ($Count=1; $Count -le $Seconds; $Count++)
243         {
244             Write-Progress -Id 1 -Activity $Message -Status "Waiting for $Seconds seconds,
$(($Seconds - $Count) left" -PercentComplete (($Count / $Seconds) * 100)
245             Start-Sleep -Seconds 1
246         }
247
248         Write-Progress -Id 1 -Activity $Message -Status "Completed" -PercentComplete 100
-Completed
249     }
250 }
251
252 function Test-Verbose
253 {
254     <#
255     .DESCRIPTION
256         The purpose of this function is to return true/false depending on whether
257         the verbose parameter has been passed to the calling PowerShell file.
258
```

```
259 .NOTES
260     Written by: Jeff Brusoe
261     Last Updated by: Jeff Brusoe
262     Last Updated: August 10, 2018
263 #>
264
265 [cmdletbinding()]
266 [OutputType([bool])]
267 param ()
268
269 Write-Output "Test-Verbose: Testing for verbose parameter" | Out-Host
270
271 if ($PSCmdlet.MyInvocation.BoundParameters["Verbose"].IsPresent)
272 {
273     Write-Output "Test-Verbose: Verbose is present" | Out-Host
274     return $true
275 }
276 else
277 {
278     Write-Output "Test-Verbose: Verbose is not present" | Out-Host
279     return $false
280 }
281 }
282
283 Function Write-ColorOutput
284 {
285     <#
286     .DESCRIPTION
287     This function allows color output in combination with Write-Output.
288     It's needed since Write-Output doesn't support this feature found in Write-Host.
289     Write-Output is used due to some issues writing log files.
290
291     In this code, ForegroundColor refers to the color of the text.
292
293     .NOTES
294     Written by: Jeff Brusoe
295     Last Updated: June 5, 2020
296     #>
297
298 [CmdletBinding(PositionalBinding=$false)]
299 param (
300     [Parameter(Mandatory=$true, ValueFromPipeline=$true)][string[]]$Message,
301     [string]$ForegroundColor = "Green"
302 )
303
304 begin
305 {
306     if ([string]::IsNullOrEmpty($Message))
307     {
308         Write-Warning "A null message value was passed into the function." | Out-Host
309         return $null
310     }
311     elseif ([Enum]::GetValues([System.ConsoleColor]) -NotContains $ForegroundColor)
312     {
313         Write-Verbose "An invalid system color was passed into the function. The default
```

```
        value of green is being used." | Out-Host
314     $ForegroundColor = "Green"
315 }
316
317 $CurrentColor = [Console]::ForegroundColor
318 $BackgroundColor = [Console]::BackgroundColor
319
320 if ($CurrentColor -eq $ForegroundColor)
321 {
322     Write-Verbose "Current color matches input foreground color." | Out-Host
323 }
324
325 if ($BackgroundColor -eq "ForegroundColor")
326 {
327     Write-Verbose "Foreground color matches background color and will not be
changed." | Out-Host
328 }
329 }
330
331 process
332 {
333     [Console]::ForegroundColor = $ForegroundColor
334
335     foreach ($m in $Message)
336     {
337         Write-Output $m
338     }
339 }
340
341 end
342 {
343     [Console]::ForegroundColor = $CurrentColor
344 }
345 }
346
347 function Get-LogFileName
348 {
349     <#
350     .SYNOPSIS
351         This function generates the names of the various log files.
352
353     .DESCRIPTION
354         The purpose of this function is to generate the names of log files used by the
calling file. It is used
355         with the Start-Transcript cmdlet. The path used is the one supplied by the
$LogFilePath parameter
356         which is being passed into the function. The date format used is Year-Month-Day(2
digit)-Hour(24 hour time)-Minute(2 digit).
357
358     .PARAMETER ProgramName
359         ProgramName is the user provided name of the program. It is used to help
360         build the session transcript log name. If it is null, then its use is omitted.
361
362     .NOTES
363         Written by: Jeff Brusoe
```

```
364     Last Updated by: Jeff Brusoe
365     Last Updated: June 3, 2020
366
367     See this link for information about PowerShell's return values and why Out-Null is
    used here.
368     https://docs.microsoft.com/en-us/powershell/module/microsoft.powershell.core/about/about_return?view=powershell-6
369     #>
370
371     [Cmdletbinding()]
372     [OutputType([string])]
373     Param(
374         [string]$ProgramName=$null,
375         [ValidateSet("SessionTranscript","Error","Output","Other")]
    [string]$LogFileType="SessionTranscript",
376         [ValidateSet("txt","csv","log")][string]$FileExtension="txt"
377     )
378
379     Write-Output "Generating $LogFileType log file name..." | Out-Null
380
381     [string]$LogFile=$null
382
383     if ([string]::IsNullOrEmpty($ProgramName))
384     {
385         $LogFile = $LogFilePath + "\" + (Get-Date -format yyyy-MM-dd-HH-mm) +
    "$LogFileType.$FileExtension"
386     }
387     else
388     {
389         $LogFile = $LogFilePath + "\" + (Get-Date -format yyyy-MM-dd-HH-mm) +
    "$ProgramName-$LogFileType.$FileExtension"
390     }
391
392     Write-Verbose "Log File: $LogFile" | Out-Host
393
394     return $LogFile
395 }
396
397 function Test-LogFilePath
398 {
399     <#
400     .DESCRIPTION
401         This function verifies that the log file path exists.
402         An option exists to create the path if it doesn't exist.
403
404     .NOTES
405         Written by: Jeff Brusoe
406         Last Updated by: Jeff Brusoe
407         Last Updated: April 10, 2018
408     #>
409
410     [Cmdletbinding()]
411     [OutputType([bool])]
412     Param(
413         [string]$LogFilePath,
```



```
414     [switch]$CreatePath
415 )
416
417 if ([string]::IsNullOrEmpty($LogFilePath))
418 {
419     Write-Warning "Log file path is empty." | Out-Null
420
421     return $false
422 }
423
424 if (!(Test-Path -Path $LogFilePath))
425 {
426     if ($CreatePath)
427     {
428         Write-Output "Log file path doesn't exist and is being created..." | Out-Null
429
430         try
431         {
432             New-Item -Path $LogFilePath -ItemType "Directory" -ErrorAction "Stop"
433             return $true
434         }
435         catch
436         {
437             Write-Error "Unable to create log file path directory" | Out-Null
438             return $false
439         }
440     }
441 }
442 else
443 {
444     return $true
445 }
446 }
447
448 Function Remove-OldLogFile
449 {
450     <#
451     .DESCRIPTION
452     This function searches for log files older than three days (or a value specified by
453     the user)
454     and removes (or copies) the files from a specified directory.
455
456     .NOTES
457     Written by Kevin Russell
458     Last updated by: Jeff Brusoe
459     Last Updated: August 26, 2019
460
461     Function Status: Working, but making changes to improve functionality.
462
463     To Do
464     1. Add ability to copy files instead of delete
465     2. Loop around if/then statements for file paths until a valid path is entered.
466     3. Add ability to accept custom file extensions
467     #>
```

```
468 [Diagnostics.CodeAnalysis.SuppressMessageAttribute("PSUseShouldProcessForStateChanging
unctions", "", Justification = "Just needed to remove old log files")]
469 [Alias("Remove-OldLogFiles")]
470 [Cmdletbinding()]
471 [OutputType([string])]
472 Param(
473     [string]$path = $($MyInvocation.PSScriptRoot + "\Logs\"),
474     [switch]$CSV,
475     [switch]$TXT,
476     [switch]$LOG,
477     [switch]$LBB, #Generated from SAN encryption key backup
478     [switch]$Delete,
479     #[string]$CopyPath = $null - Needs to be implemented
480     [int]$Days = 3
481 )
482
483 Write-Verbose "Days to keep log files: $Days"
484
485 if ($Days -gt 0)
486 {
487     $Days = -1*$Days
488 }
489
490 if ($Delete)
491 {
492     Write-Output "Files will be deleted."
493 }
494 else
495 {
496     Write-Output "Files will not be deleted."
497 }
498
499 $time = (Get-Date).AddDays($Days)
500
501 Write-Verbose "Removing old log files"
502
503 $ValidPath = $false
504
505 while(!$ValidPath)
506 {
507     if ([string]::IsNullOrEmpty($path))
508     {
509         $path = Read-Host "Please enter the directory path"
510     }
511     elseif (!(Test-Path $path))
512     {
513         $path = Read-Host "Please enter a valid directory path"
514     }
515     else
516     {
517         $ValidPath = $true
518     }
519 }
520
```

```
521 $RemoveString = @() #Array of file extensions to remove
522
523 if ($CSV)
524 {
525     Write-Verbose "Adding csv files to remove string."
526     $RemoveString += "*.csv"
527 }
528
529 if ($TXT)
530 {
531     Write-Verbose "Adding txt files to remove string"
532     $RemoveString += "*.txt"
533 }
534
535 if ($LOG)
536 {
537     $RemoveString += "*.log"
538 }
539
540 if ($LBB)
541 {
542     $RemoveString += "*.lbb"
543 }
544
545 if (($RemoveString | Measure-Object).Count -eq 0)
546 {
547     Write-Output "No files to remove"
548
549     return $null
550 }
551
552 Write-Verbose "RemoveString: $RemoveString"
553
554 if ([string]::IsNullOrEmpty($RemoveString))
555 {
556     Write-Output "Unable to remove any files."
557 }
558 else
559 {
560     Write-Output "Path: $path"
561     $files = Get-ChildItem -path $path\* -Include $RemoveString
562
563     if ($null -eq $files)
564     {
565         #Nothing is found
566         Write-Verbose "No files in directory"
567     }
568     else
569     {
570         Write-Verbose $("File Count: " + ($files | Measure-Object).Count)
571
572         foreach ($file in $files)
573         {
574             #Write-Output $file.FullName
575         }
576     }
577 }
```

```
576         if($file.LastWriteTime -lt $time)
577         {
578             if (!$Delete)
579             {
580                 Write-Output $("Potential Delete: " + $file.FullName)
581             }
582             else
583             {
584                 Write-Verbose $("Removing: " + $file.FullName)
585
586                 Remove-Item -Path $file.fullname -Force
587             }
588         }
589     }
590 }
591 }
592 }
593
594 Function Write-LogFileSummaryInformation
595 {
596     <#
597     .DESCRIPTION
598         This function writes common information to log files used for Active Directory
599         and Exchange PowerShell files.
600
601     .NOTES
602         Written By: Matt Logue
603         Last Updated: November 13, 2016
604     #>
605
606     [cmdletbinding()]
607     Param(
608         [string]$FilePath = $null, #A null path will just put this information on the
screen
609         [switch]$ComputerName, #$true = include computer name in log file
610         [switch]$ExcludedUsers, #$true = display list of users excluded from processing
611         [string]$Summary = $null #if not null output summary
612     )
613
614     $dateTime = Get-Date -Format G
615
616     if (([string]::IsNullOrEmpty($FilePath)) -or ((Test-Path -Path $FilePath) -eq
>false))
617     {
618         Write-Verbose $("*----- "+$dateTime+"-----*") | Out-Host
619         Write-ColorOutput -Message "File Path is Empty" -ForegroundColor "Green"
-Verbose | Out-Host
620
621         if ($ComputerName -eq $true)
622         {
623             Write-Verbose $("Computer Name: " + $env:computername) | Out-Host
624         }
625
626         if ($ExcludedUsers -eq $true)
627         {
```

```
628     Write-Verbose $("Excluded Users: ") | Out-Host
629 }
630
631 if (![string]::IsNullOrEmpty($Summary))
632 {
633     Write-Verbose $("Summary: " + $Summary) | Out-Host
634 }
635
636 }
637 else
638 {
639
640     Write-Verbose ("*----- $dateTime -----*`n`r") | Out-Host
641     Add-Content -Value ("*----- $dateTime -----*`n`r") -Path
$FilePath
642
643     Write-Verbose $("File: " + $FilePath) | Out-Host
644     Add-Content -Value "File: $FilePath`n`r" -Path $FilePath
645
646     if ($ComputerName -eq $true)
647     {
648         Write-Verbose $("Computer Name: " + $env:computername) | Out-Host
649         Add-Content -Value "`r`nComputerName: $env:computername`n`r" -Path $FilePath
650     }
651
652     if ($ExcludedUsers -eq $true)
653     {
654         Write-Verbose $("Excluded Users: ") | Out-Host
655         Add-Content -Value "`r`nExcluded Users: `r`n" -Path $FilePath
656     }
657
658     if (![string]::IsNullOrEmpty($Summary))
659     {
660         Write-Verbose $("Summary: " + $Summary) | Out-Host
661         Add-Content -Value "Summary:`r`n$Summary" -Path $FilePath
662     }
663 }
664 }
665
666 Function Send-Email
667 {
668     <#
669     .DESCRIPTION
670     The purpose of this function is to serve as a wrapper for the Send-MailMessage
cmdlet. This is done to handle
671     decrypting the encrypted password file which is needed to relay mail with Send-
MailMessage.
672
673     .NOTES
674     Written by: Jeff Brusoe
675     Last Updated by: Jeff Brusoe
676     Last Updated: April 16, 2018
677
678     This function probably isn't needed anymore. Need to see if it is still being used
before removing it though.
```

```
679  #>
680
681  [CmdletBinding()]
682  Param (
683      [string[]]$To,
684      [string]$From,
685      [string]$Subject,
686      [string]$MessageBody,
687      [string[]]$Attachments,
688      [string]$SMTPServer = "Hssmtp.hsc.wvu.edu"
689  )
690
691  Write-Verbose "Preparing to send email..." | Out-Host
692  $Error.Clear()
693
694  try
695  {
696      Send-MailMessage -to $To -From $From -SMTPServer $SMTPServer -Subject $Subject
-UseSSL -port 587 -Attachments $Attachments -Body $MessageBody -ErrorAction Stop
697  }
698  catch
699  {
700      Write-Warning "Unable to send email message" | Out-Host
701  }
702 }
703
704 function Get-PasswordFromSecureStringFile
705 {
706
707     <#
708     .DESCRIPTION
709         The purpose of this function is to decrypt a secure string file to handle user
authentication to
710         AD or Office 365.
711
712     .NOTES
713         Written by: Jeff Brusoe
714         Last Updated by: Jeff Brusoe
715         Last Updated: August 27, 2019
716     #>
717
718     [CmdletBinding()]
719     param (
720         [bool]$Prompt=$false,
721         [bool]$ChangeSecureStringFile=$false,
722         [string]$PWFile = ".\EncryptedPassword.txt" #Mandatory
723     )
724
725     [string]$Password=$null
726
727     if ($ChangeSecureStringFile)
728     {
729         try
730         {
731             Read-Host "Enter Current Password" -AsSecureString | convertfrom-securestring |
```

```
Out-File $PWFile
732     Write-ColorOutput -foregroundcolor "Green" -Message "Successfully updated secure
string file.`n" | Out-Host
733 }
734 catch
735 {
736     $Prompt = $false
737     Write-Error "There was an error generating the secure string file.`n" | Out-Host
738 }
739 }
740
741 if ($Prompt)
742 {
743     $Password = Read-Host "Enter Password" | Out-Host
744 }
745 else
746 {
747     if (Test-Path $PasswordFile)
748     {
749         Write-ColorOutput -ForegroundColor "Green" -Message "Decrypting Password..." |
Out-Host
750
751         try
752         {
753             $securestring = convertto-securestring -string (get-content $PWFile)
754             $bstr =
[System.Runtime.InteropServices.Marshal]::SecureStringToBSTR($securestring)
755             $Password = [System.Runtime.InteropServices.Marshal]::PtrToStringAuto($bstr)
756
757             Write-ColorOutput -ForegroundColor "Green" -Message "Password decrypted
successfully." | Out-Host
758         }
759         catch
760         {
761             Write-Error "There was an error decrypting the password. Exiting file." | Out-
Host
762         }
763     }
764 }
765
766 Return $Password
767 }
768
769 function Get-RandomPassword
770 {
771     <#
772     .DESCRIPTION
773     The purpose of this function is to generate a random password. The password
generated meets
774     WVU password complexity requirements:
775     1. Must be between 8 and 20 characters in length.
776     2. Must contain characters from at least three of the following four categories:
777         a. Uppercase letters: A-Z
778         b. Lowercase letters: a-z
779         c. Numbers: 0-9
```

```
780         d. Only these special characters: ! ^ ? : . ~ - _
781
782     .NOTES
783         Written by: Jeff Brusoe
784         Last Updated by: Jeff Brusoe
785         Last Updated: April 12, 2018
786     #>
787
788     [CmdletBinding()]
789     [OutputType([string])]
790     param (
791         [int]$PasswordLength = 19
792     )
793
794     Write-Output "Generating random password for new AD account" | Out-Host
795
796
797     Write-Output "Generating password for new AD account" | Out-Host
798
799     #https://blogs.technet.microsoft.com/undocumentedfeatures/2016/09/20/powershell-
800     random-password-generator/
801     [string]$Password = ([char[]]([char]33..[char]95) + ([char[]]([char]97..[char]126))
802     0..9 | Sort-Object {Get-Random})[0..$PasswordLength] -join ''
803
804     Write-Output "Password: $Password" | Out-Host
805
806     return $Password
807 }
808
809 function Exit-Command
810 {
811     <#
812     .DESCRIPTION
813         This function is called to handle error conditions where a PS file should exit.
814
815     .NOTES
816         Written by: Jeff Brusoe
817         Last Updated by: Jeff Brusoe
818         Last Updated: June 4, 2020
819     #>
820
821     [CmdletBinding()]
822     [Alias("Exit-Commands")]
823     param ()
824
825     #To do: Display way program is stopping (Complete, error & location, etc.)
826     try
827     {
828         Stop-Transcript -ErrorAction Stop
829     }
830     catch
831     {
832         Write-Verbose "Unable to stop transcript"
```



```
833 Exit
834 }
835
836 function Test-ValidWVUEmail
837 {
838     <#
839     .DESCRIPTION
840         This function tests whether an email is a valid WVU email address. It only checks
841         if
842         it's possible, but not that the account actually exists.
843     .NOTES
844         Written by: Jeff Brusoe
845         Last Updated by: Jeff Brusoe
846         Last Updated: June 8, 2020
847     #>
848
849     [CmdletBinding()]
850     [OutputType([bool])]
851     param (
852         [Parameter(Mandatory=$True)][string]$EmailAddress
853     )
854
855     $ValidEmail = $false
856
857     Write-Verbose "Attempting to verify: $EmailAddress" | Out-Host
858
859     if (($EmailAddress.IndexOf("wvu.edu") -gt 0) -AND ($EmailAddress.IndexOf("@") -gt 0))
860     {
861         Write-Verbose "The email is valid" | Out-Host
862         $ValidEmail = $true
863     }
864     else
865     {
866         Write-Verbose "The email is invalid" | Out-Host
867     }
868
869     return $ValidEmail
870 }
871
872 #####
873 # Export functions #
874 #####
875
876 Export-ModuleMember -Function "Get-*"
877 Export-ModuleMember -Function "Set-Environment"
878 Export-ModuleMember -Function "Write-*"
879 Export-ModuleMember -Function "Send-Email"
880 Export-ModuleMember -Function "Set-WindowTitle"
881 Export-ModuleMember -Function "Remove-OldLogFile" -Alias "Remove-OldLogFiles"
882 Export-ModuleMember -Function "Start-*"
883 Export-ModuleMember -Function "Test-*"
884 Export-ModuleMember -Function "Exit-Command" -Alias "Exit-Commands"
```

