```
################################################################################
                ############ START ITNG CUSTOM LOGIN SCRIPT ############
################################################################################

<# Trying out new method using a batch file labeled "RUNME.bat" in the flash drive's Scripts folder for easy start.
If batch script does NOT work, you must run this command in admin powershell separately first. THEN you can run the rest of this script as a single powershell script execution.
>>> Set-ExecutionPolicy -ExecutionPolicy Bypass -Scope Process -Force
    <#Note: This MUST be entered manually first, or it will not allow this script to be run. You can run this in Powershell ISE as an administrator and run it all at once.
    >>> Make sure you have the scripts folder with all packages and installers listed below in this script for it to run properly. <<< #>


################################################################################
                ###     CREATE LOG FILE SECTION <START>     ###
################################################################################
#region

<#NOTES for USB drive:
    : make sure you have all the necessary installers in the source folder on your USB drive
        : and that your USB drive is listd as D: or change it as necessary.
        : otherwise this script will fail to start.
    : you must use "Scripts" as your folder name in your USB drive, and not "sources".
        : "Sources" is already a folder in the USB installer as a separate folder and will not work here.
#>

            ###CREATE LOCAL SOURCES FOLDER FOR INSTALLATION AND LOGGING.###


#define folders for holding the installers, scripts, and log files.
$sourceFolder = "D:\Scripts"
$destinationFolder = "C:\Sources"
#CRITICAL NOTE: If copy function fails: FORCE STOP rest of this script automatically.
try {
# First checks if there is a sources folder on C: and creates new folder if it doesn't exist.
    if (-not (Test-Path $destinationFolder)) {
    # Create "C:\Sources\" directory to store log files &&  additional applications and scripts as needed.
        New-Item -Path $destinationFolder -ItemType Directory
        Write-Host "Created new 'Sources' folder at: $destinationFolder" -ForegroundColor Green
    } else {
        Write-Host "$destinationFolder already exists." -ForegroundColor Yellow
    }

# Second checks if the USB installer source folder exists
    if (Test-Path $sourceFolder) {
    # Copy the folder's subcontents to new local sources folder:
        Copy-Item -Path "$sourceFolder\*" -Destination $destinationFolder -Recurse -Force -ErrorAction Stop
        Write-Host "Copied all content from $sourceFolder folder to $destinationFolder" -ForegroundColor Yellow
    } else {
    # throw critical error and force TERMINATE the rest of this script if the folder copy fails:
    Write-Host "CRITICAL ERROR:" -ForegroundColor Red
    Write-Host "$sourceFolder does not exist or cannot be found.  `
        This source folder MUST exist as '$sourceFolder' on your USB drive `
        in order to copy over necessary files to the new computer. `
        This script will throw multiple errors if the source and destination folders do not exist." -ForegroundColor Red
    throw "Script function has terminated due to CRITICAL ERROR listed above. Please correct CRITICAL ERROR and rerun script."
    }
```

```
} catch {
    # Log the error and force stop the script
    Write-Host "ERROR: $($_.Exception.Message)" -ForegroundColor Red
    exit 1  # Force stop the entire script with an error code
}
#log success and proceed with script.
Write-Host "No errors when creating required folder: '$destinationFolder'." -ForegroundColor Green
Write-Host "STARTING MAIN SCRIPT NOW!" -ForegroundColor Cyan

    <# NOTES
        : If the script reaches this point, the copy was successful, and the rest of the script can continue.
        : This script WILL terminate itself if the above function was not succesful.
        : You MUST ensure both the above folder paths are named and set correctly for this script to work.
        : Fix any folder issues and rerun script from the beginning.
    #>


            ###CREATE LOG FILE TO REVIEW AFTER SCRIPT COMPLETES.###


# Define log file as variable
    $logFile = "C:\Sources\initial_setup_log.txt"   # Change TXT filename as needed!
# Create log file using the variable defined above.
if (Test-Path $logFile) {
        Write-Host "Log file already exists at $logFile." -ForegroundColor Yellow
} else {
        New-Item -ItemType File -Path $logFile -Force | Out-Null
        Write-Host "New setup log file created at $logFile." -ForegroundColor Green
}
Write-Host "Remember to check your setup log file: '$logFile' after reboot to see what has been done!" -ForegroundColor Cyan


            ###CREATE CUSTOM FUNCTION TO LOG OUTPUT MESSAGES IN THIS SCRIPT.###

# Create function to log messages to the log file.
function Log-Message {
# define paramater to use the "output" string in the function:
    param ( [string]$message )
    # note:  this is defined as the "[string]" in each call of the Log-Message function.
# create time stamp for each log entry:
    $timestamp = Get-Date -Format "yyyy-MM-dd HH:mm:ss"
# define a variable to create log entry with message output and time stamp:
    $logEntry = "$timestamp - $message"
# create and append new message(s) in log file:
    Add-Content -Path $logFile -Value $logEntry
}

# First log message.
    Log-Message "Script execution started."

    <# NOTES on log message
        : Function Name is Case Sensitive!
        : Will log the string as text you manually define in "" with time stamp into the TXT log file.
    #>


################################################################################
                    ###     CREATE LOG FILE SECTION <END>       ###
################################################################################
#endregion


################################################################################
                    ###     MANUAL ENTRY REQUIRED SECTION <START>       ###
```

```powershell
################################################################################################
#region


            ###CHANGE COMPUTER NAME WITH MANUAL INPUT.###


# Prompt the user to enter the new computer name in CLI and store as variable.
    $newName = Read-Host "Please enter the new computer name"
# Get the current computer name and store as variable.
    $currentName = $env:COMPUTERNAME
# Print to screen the current and new computer name
    Write-Host "Current Computer Name: $currentName" -ForegroundColor Yellow
    Write-Host "New Computer Name will be: $newName" -ForegroundColor Green
# Command to change the computer name
    Rename-Computer -NewName $newName -Force
# Print to screen the PC name change confirmation and log in log file.
    Write-Host "Computer name changed successfully to '$newName'" -ForegroundColor Green
    Log-Message "Computer name changed successfully to '$newName'"


################################################################################################
                    ###     MANUAL ENTRY REQUIRED SECTION <END>     ###
################################################################################################
#endregion


################################################################################################
                    ###     BEGIN MAIN SCRIPT FUNCTION SECTION <START>      ###
################################################################################################
#region

                            # working on this...


################################################################################################
                    ###     <END>   ###
################################################################################################
#endregion

################################################################################################
                    ###     NETWORKING SETTINGS SECTION <START>     ###
################################################################################################
#region


            ####DISABLE IPV6 ON ALL ADAPTERS.####

try {
# Get all active (Up) network adapters
    $adapters = Get-NetAdapter | Where-Object { $_.Status -eq 'Up' }
    foreach ($adapter in $adapters)
    {# Loop through each adapter to disable IPv6
    Write-host "Disabling IPv6 on all network adapters." -ForegroundColor Yellow
# Disable IPv6 on network adapter
    Set-NetAdapterBinding -Name $adapter.Name -ComponentID ms_tcpip6 -Enabled $false -
    ErrorAction Stop
    }
# Log success after processing all adapters
    Write-host "Disabled IPv6 on all network adapters." -ForegroundColor Green
    Log-Message "IPv6 has been disabled on the adapter: $($adapter.Name)."
}
catch {
# Log failure(s).
    Write-Host "ERROR: Failed to disable IPv6 on the adapter: $($adapter.Name). Error:
    $_" -ForegroundColor Red
    Log-Message "ERROR: Failed to disable IPv6 on the adapter: $($adapter.Name). Error:
    $_"
}
```

```powershell
# Log IPv6 disable completion.
    Log-Message "IPv6 disable operation completed for all network adapters."


            ###SET NETWORK TYPE TO PRIVATE FOR ALL ADAPTERS (DEFAULT IS PUBLIC).###


try {
# Get all network adapters with a network connection
    $networkAdapters = Get-NetConnectionProfile
# Loop through each network adapter
    foreach ($adapter in $networkAdapters)
    {# Check if the adapter is Ethernet or Wi-Fi
        if ($adapter.InterfaceAlias -like "*Ethernet*" -or $adapter.InterfaceAlias -like "*Wi-Fi*")
        {# Set network profile to private for the adapter
            Write-Host "Setting network profile for $($adapter.Name) ($($adapter.InterfaceAlias)) to Private." -ForegroundColor Yellow
            Set-NetConnectionProfile -InterfaceIndex $adapter.InterfaceIndex -NetworkCategory Private
        }}
# Log success after processing all adapters
    Write-Host "All Ethernet and Wi-Fi network profiles set to Private." -ForegroundColor Green
    Log-Message "All Ethernet and Wi-Fi network profiles set to Private."
}
catch {
# Log failure if an error occurs
    Write-Host "An error occurred while setting network profiles: $($_.Exception.Message)" -ForegroundColor Red
    Log-Message "An error occurred while setting network profiles: $($_.Exception.Message)"
}


            ### ENABLE ALL WINDOWS FIREWALLS.###

try {

    Set-NetFirewallProfile -Profile Public, Domain, Private -Enabled True
    Write-Host "Enabled ALL Firewalls" -ForegroundColor Green
    Log-Message "Enabled ALL Firewalls"
}
catch {
    # Log any failures
    Write-Host "FAILED to enable all Windows firewalls. Error: $_" -ForegroundColor Red
    Log-Message "FAILED to enable all Windows firewalls. Error: $_"
}


            ### ADD WINDOWS FIREWALL RULE TO ALLOW RDP WITH FIREWALL ON.###
try {
    Write-Host "Configuring Windows Firewall to allow RDP (port 3389) on Private and Domain profiles..." -ForegroundColor Yellow
# Enable Remote Desktop rules only for Private and Domain profiles
    Get-NetFirewallRule -DisplayGroup "Remote Desktop" | Where-Object { $_.Profile -match 'Domain|Private' } | Enable-NetFirewallRule
# Explicitly open port 3389 for TCP, only on Private and Domain profiles
    New-NetFirewallRule -DisplayName "Allow RDP Port 3389" `
        -Direction Inbound `
        -LocalPort 3389 `
        -Protocol TCP `
        -Action Allow `
        -Profile Domain,Private `
        -ErrorAction SilentlyContinue
    # Log port rule addition
    Write-Host "Created new firewall rule to allow Port 3389 (RDP) on Private and Domain profiles in Windows Firewall." -ForegroundColor Green
    Log-Message "Created new firewall rule to allow Port 3389 (RDP) on Private and
```

```
        Domain profiles in Windows Firewall."
245 } catch {
246     Write-Host "Failed to add firewall rule to allow RDP over port 3389 for private and
        domain networks. Error: $_" -ForegroundColor Red
247     Log-Message "Failed to add firewall rule to allow RDP over port 3389 for private and
        domain networks. Error: $_"
248 }
249
250             ### CONFIGURE REMOTE ACCESS ####
251
252 # ENABLE RDP CONNECTIONS
253     New-ItemProperty -Path "HKLM:\System\CurrentControlSet\Control\Terminal Server" -Name
         "fDenyTSConnections" -PropertyType DWORD -Value 0 -Force
254 # REQUIRE NETWORK LEVEL AUTHENTICATION
255     New-ItemProperty -Path "HKLM:\System\CurrentControlSet\Control\Terminal
        Server\WinStations\RDP-Tcp" -Name "UserAuthentication" -PropertyType DWORD -Value 1 -
        Force
256 # Log changes
257     Write-Host "Enabled RDP connections with network level authentication required" -
        ForegroundColor Green
258     Log-Message "Enabled RDP connections with network level authentication required"
259 # Log RDP completion
260     Log-Message "RDP configuration completed. Check above log messages to confirm status
        and any errors."
261
262
263
264 ##############################################################################################
265                     ###     NETWORKING SETTINGS SECTION <END>   ###
266 ##############################################################################################
267 #endregion
268
269 ##############################################################################################
270                     ###     POWER SETTINGS SECTION <START>      ###
271 ##############################################################################################
272 #region
273
274
275 ###SET ACTIVE POWER PLAN:####
276
277     # Set the power scheme to High Performance (predefined GUID)
278 powercfg.exe /setactive 8c5e7fda-e8bf-4a96-9a85-a6e23a8c635c
279     # Disable sleep on AC and DC (battery) power
280 powercfg -x standby-timeout-ac 0        # Disables sleep when on AC power
281 powercfg -x standby-timeout-dc 0        # Disables sleep when on battery power
282     # Set the display to turn off after 20 minutes on both AC and DC power
283 powercfg -x monitor-timeout-ac 20       # Turns off display after 20 minutes on AC power
284 powercfg -x monitor-timeout-dc 20       # Turns off display after 20 minutes on battery
    power
285     # Disable hibernate on both AC and DC power
286 powercfg -x hibernate-timeout-ac 0      # Disables hibernate when on AC power
287 powercfg -x hibernate-timeout-dc 0      # Disables hibernate when on battery power
288
289 ##############################################################################################
290                     ### POWER SETTINGS SECTION <END> ###
291 ##############################################################################################
292 #endregion
293
294
295
296 ##############################################################################################
297                     ### MISC SECTION <START> ###
298 ##############################################################################################
299 #region
300
301 ####SET SYSTEM STARTUP ENTRIES.####
302 try {
303 # Set the boot menu timeout to 5 seconds (gives us time to enter BIOS easily)
304     bcdedit.exe /timeout 5
```

```
# Log success
    Write-Host "Boot menu timeout successfully set to 5 seconds." -ForegroundColor Green
    Log-Message "Boot menu timeout successfully set to 5 seconds."
}
catch {
# Log failure
   Write-Host "ERROR: Failed to set the boot menu timeout. Error: $_" -ForegroundColor
   Red
   Log-Message "ERROR: Failed to set the boot menu timeout. Error: $_"
}
<# NOTES: the boot loader menu will display for 5 seconds before loading boot manager #>


#### INSTALL WMIC.EXE COMMAND. ###

try {
# Displaying initial install message
    Write-Host "Installing WMIC.exe..." -ForegroundColor Yellow
# Install WMIC.exe (deprecated by Microsoft):
    Add-WindowsCapability -Online -Name WMIC~~~~ -ErrorAction Stop
} catch {
# Log unexpected errors:
    Write-Host "FAILED to install WMIC.EXE. Error: $($_.Exception.Message)" -
    ForegroundColor Red
    Log-Message "FAILED to install WMIC.EXE. Error: $($_.Exception.Message)"
}
# Log install status of WMIC.exe if no exception occurred:
$WMICpath = "C:\Windows\System32\wbem\wmic.exe"
if (Test-Path $WMICpath) {
        Write-Host "Installed WMIC.exe successfully." -ForegroundColor Green
        Log-Message "Installed WMIC.EXE successfully."
    } else {
        Write-Host "Failed to install WMIC.exe." -ForegroundColor Red
        Log-Message "Failed to install WMIC.exe. You will need to try reinstall on next
        reboot."
    }


<# notes on using dism to install wmic.exe:
    : We have RMM components and scripts that still use wmic.exe instead of the newer
    powershell versions.
    : WMIC has been deprecated and is no longer installed if the OS is later than
    windows 10 version 1809 (21H1).
    : Dism will NOT install wmic.exe using LegacyComponents.
    : All wmic.exe commands are replaced with powershell commands in this script.
#>


# ENABLE AUTOMATIC REBOOT AFTER SYSTEM FAILURE
try {
    Set-ItemProperty -Path "HKLM:\SYSTEM\CurrentControlSet\Control\CrashControl" -Name
    "AutoReboot" -Value 1 -ErrorAction Stop
#Log success
    Write-Host "AutoReboot after system failure has been successfully enabled." -
    ForegroundColor Green
    Log-Message "AutoReboot after system failure has been successfully enabled."
}
catch { # Log failure
    Write-Host "ERROR: Failed to enable AutoReboot after system failure. Error: $_" -
    ForegroundColor Red
    Log-Message "ERROR: Failed to enable AutoReboot after system failure. Error: $_"
}


    ###SET DEBUGGING INFORMATION TYPE TO NONE####
try {
    Set-ItemProperty -Path "HKLM:\SYSTEM\CurrentControlSet\Control\CrashControl" -Name
    "CrashDumpEnabled" -Value 0 -ErrorAction Stop
# Log success
```

```
    Write-Host "Debugging information type has been set to None." -ForegroundColor Green
    Log-Message "Debugging information type has been set to None."
}
catch { # Log failure
    Write-Host "ERROR: Failed to set DebugInfoType to None. Error: $_" -ForegroundColor Red
    Log-Message "ERROR: Failed to set DebugInfoType to None. Error: $_"
}


    ####UPDATE AV DEFINITIONS####
try {
    Write-Host "Updating Windows Defender Antivirus Definitions" -ForegroundColor Yellow
#update windows defender with Powershell
    Update-MpSignature
#log success
    Write-Host "AV signature definitions update completed successfully" -ForegroundColor Green
    Log-Message "AV signature definitions update completed successfully"
}
catch {
    Write-Host "ERROR: Failed to update AV signature definitions. Error: $_" -ForegroundColor Red
    Log-Message "ERROR: Failed to update AV signature definitions. Error: $_"
}


    ####SET TIMEZONE TO EASTERN TIME####
try {
#set time zone
    $ESTzone = "Eastern Standard Time"
    Set-TimeZone -Id "$ESTzone"
#log success
    Write-Host "Time zone has been set to $ESTzone" -ForegroundColor Green
    Log-Message "Time zone has been set to $ESTzone"
#update variable for time resync
    $NewTimeZone = $ESTzone
} catch {
# Log failure
    Write-Host "ERROR: Failed to set time zone to $ESTzone. Error: $_" -ForegroundColor Red
    Log-Message "ERROR: Failed to set time zone to $ESTzone. Error: $_"
}

<#      #Set TimeZone to Central Time (only used for select clients)
        #- uncomment this and comment out EST section as needed.
try {
#set time zone
    $CSTzone = "Central Standard Time"
    Set-TimeZone -Id "$CSTzone"
#log success
    Write-Host "Time zone has been set to $"$CSTzone" -ForegroundColor Green
    Log-Message "Time zone has been set to $"$CSTzone"
#update variable for time resync
    $NewTimeZone = $CSTzone
} catch {
# Log failure
    Write-Host "ERROR: Failed to set time zone to $"$CSTzone. Error: $_"
    -ForegroundColor Red
    Log-Message "ERROR: Failed to set time zone to $"$CSTzone. Error: $_"
}
#>


####RESYNC TIME CLOCK####


try {
    Write-Host "Resyncing time service to $NewTimeZone." -ForegroundColor Yellow
```

```powershell
# Stop the w32time service and wait for it to stop
    Stop-Service w32time
# Wait until the service has stopped
    while ((Get-Service w32time).Status -ne 'Stopped') {
        Start-Sleep -Seconds 1
    } Write-Host "w32time service stopped." -ForegroundColor Yellow
# Start the w32time service
    Start-Service w32time
# Wait until the service has started
    while ((Get-Service w32time).Status -ne 'Running') {
        Start-Sleep -Seconds 1
    } Write-Host "w32time service started." -ForegroundColor Yellow
# Resync the time with w32tm
    w32tm /resync /Force
    start-Sleep -Seconds 2
# Verify synchronization status
    $syncStatus = w32tm /query /status
# Check if resync was successful
    if ($syncStatus -match "Last Successful Sync Time") {
        Write-Host "Successfully resynchronized the time service to the correct time in the $NewTimeZone." -ForegroundColor Green
        Log-Message "Successfully resynchronized the time service to the correct time in the $NewTimeZone."
    } else {
        Write-Host "FAILED to resynchronize the time service to the correct time  in the $NewTimeZone." -ForegroundColor Red
        Log-Message "FAILED to resynchronize the time service to the correct time  in the $NewTimeZone. Check clock configuration."
    }
} catch {
    Write-Host "FAILED to resynchronize the time service to the correct time in the $NewTimeZone." -ForegroundColor Red
    Log-Message "FAILED to resynchronize the time service to the correct time in the $NewTimeZone. Check clock configuration."
}



################################################################################################
                    ### MISC SECTION <END> ###
################################################################################################
#endregion

################################################################################################
                    ### APPLICATION INSTALLATION SECTION <START>###
################################################################################################
#region

#####Install 'NuGet' package if missing from system, depending on Windows version.#####

try {
# Suppress any confirmation prompts
    $ConfirmPreference = 'None'
# install package
    Write-Host "Installing latest version of 'NuGet' package from Microsoft" -ForegroundColor Yellow
    Install-PackageProvider -Name NuGet -MinimumVersion 2.8.5.201 -Force
# log success
    Write-Host "SUCCESS: Installed latest version of 'NuGet' package from Microsoft" -ForegroundColor Green
    Log-Message "Installed latest version of 'NuGet' package from Microsoft"
}
catch {
# log errors (if any)
    Write-Host "ERROR: Failed to install NuGet. Error: $($_.Exception.Message)" -ForegroundColor Red
    Log-Message "ERROR: Failed to install NuGet. Error: $($_.Exception.Message)"
}

```

```
489
490             ###RUN WINGET TO INSTALL STANDARD APPLICATIONS.###
491
492 Write-Host "Running 'WinGet' to install standard software applications." -ForegroundColor
     Yellow
493
494 # Define the applications to install (name and corresponding winget package)
495 $apps = @(
496     @{Name="Powershell 7"; Package="microsoft.powershell"},
497     @{Name="Google Chrome"; Package="Google.Chrome"},
498     @{Name="Adobe Acrobat Reader"; Package="adobe.acrobat.reader.64-bit"},
499     @{Name="Dell Command Update"; Package="Dell.CommandUpdate"},
500 #   @{Name="Splashtop Streamer"; Package="Splashtop.SplashtopStreamer"},
501 #    @{Name="VLC Media Player"; Package="VideoLAN.VLC"}
502 #    @{Name="Firefox"; Package="Mozilla.Firefox"}
503 # Uncomment and add more applications as needed:
504     # @{Name=" "; Package=" "},
505     # @{Name=" "; Package=" "},
506     # @{Name=" "; Package=" "},
507     # @{Name=" "; Package=" "},
508     # @{Name=" "; Package=" "},
509 )
510 <#NOTES:
511     : make sure there is a "," after each "}" until the last "}" to ensure all apps are
        installed.
512     : you need to use the specific package name as listed in the winget repository
513     : winget search "*app name*" will return list of all available versions of the
        application.
514         : The * * act as wild cards for your query.
515 #>
516
517
518 ## INSTALL ALL APPLICATIONS LISTED ABOVE.##
519 foreach ($app in $apps) { #attempt to install all applications listed in above function
520 try {
521 # Install the application silently using winget:
522     Write-Host "Installing $($app.Name)..." -ForegroundColor Yellow
523     winget.exe install $app.Package --scope machine --silent --accept-source-agreements
524         # Note: will still show installation progress bar in CLI.
525 # log success
526     Write-Host "Installed $($app.Name) successfully." -ForegroundColor Green
527     Log-Message "Installed $($app.Name) successfully."
528 } catch {
529 # Log any failures
530     Write-Host "Failed to install $($app.Name). Error: $_" -ForegroundColor Red
531     Log-Message "ERROR: Failed to install $($app.Name). Error: $_"
532       }
533 }
534
535 ## update all installed software through WinGet
536     Write-Host "Updating all apps installed by Winget." -ForegroundColor Yellow
537     winget.exe upgrade --all
538 # log success
539     Write-Host "All winget-based software upgrades installed successfully." -
        ForegroundColor Green
540     Log-Message "All winget-based software upgrades installed successfully."
541
542 <# NOTES:
543     : for quick single installs use the standard installation command below.
544         : Add app package name in the "".
545     : If you do not include the --silent switch, it will give you verbose installation
        progress by default.
546     : The --accept-source-agreements is used to auto select "yes" to use the ms store
        and allow the command to run automatically.
547
548 #winget.exe install "" --scope machine --accept-source-agreements
549 #winget.exe install "" --scope machine --accept-source-agreements
550 #winget.exe install "" --scope machine --accept-source-agreements
551 #winget.exe install "" --scope machine --accept-source-agreements
```

```powershell
#winget.exe install "" --scope machine --accept-source-agreements

#>

#####INSTALL PS MODULE TO ALLOW POWERSHELL 7 TO RUN WINDOWS UPDATE.#####
try {
# Install PS module to allow Powershell 7 to run Windows Update.
    Write-Host "Installing Powershell Module 'PSWindowsUpdate' to enable Windows Updates through Powershell" -ForegroundColor Yellow
    Install-Module PSWindowsUpdate -Force
# log success
    Write-Host "Installed Powershell Module 'PSWindowsUpdate' to enable Windows Updates through Powershell" -ForegroundColor Green
    Log-Message "Installed Powershell Module 'PSWindowsUpdate' to enable Windows Updates through Powershell"
    }
catch {
# log errors (if any)
    Write-Host "ERROR: Failed to install the Powershell Module 'PSWindowsUpdate' to enable Windows Updates through Powershell Error: $_" -ForegroundColor Red
    Log-Message "ERROR: Failed to install the Powershell Module 'PSWindowsUpdate' to enable Windows Updates through Powershell Error: $_"
    }


## INSTALL OFFICE 365 ##

#define sources directory as a variable - change as needed.
    $sources = "C:\Sources"
#set file path that contains officesetup.exe installer
    $Office365InstallPath = "$sources\OfficeSetup.exe"
# Specify the configuration file path - necessary for silent install
    $configurationFilePath = "$sources\O365Configuration.xml"
# Set the arguments to include the /configure switch
    $arguments = "/configure $configurationFilePath"
# Set variable to start officesetup.exe with arguments and configuration file.
    $process = Start-Process -FilePath $Office365InstallPath -ArgumentList $arguments -PassThru
        #note: no "-Wait" needed here as we're using -PassThru
try {
    Write-Host "Starting Office 365 installation in the background. `
    This will take a few minutes to install and will automatically move onto the next step." -ForegroundColor Cyan
    Log-Message "Started Office 365 installation."
# START INSTALL process and wait for the process to complete
    $process.WaitForExit()
# Capture the exit code from the process object
    $exitCode = $process.ExitCode
# Check if installation was successful
    if ($exitCode -eq 0) {
    # log if successful
        Write-Host "Office 365 installed successfully." -ForegroundColor Green
        Log-Message "Office 365 installed successfully."
    } else {
    # log errors
        Write-Host "Office 365 installation failed with exit code: $exitCode" -ForegroundColor Red
        Log-Message "Office 365 installation failed with exit code: $exitCode"
    }
} catch {
    # log failure to install.
    $errorMessage = $_.Exception.Message
    Write-Host "Error during Office 365 installation: $errorMessage" -ForegroundColor Red
    Log-Message "Error during Office 365 installation: $errorMessage"
}



<# install Sonicwall NetExtender:
```

```
613
614 msiexec.exe /i "D:\Scripts\NetExtender-x64-10.2.341.msi" /qn /norestart server=#.#.#.#
    domain=LocalDomain EDITABLE=TRUE netlogon=true ALLUSERS=2
615     <# notes:
616     : /qn = silent install
617     : /norestart = does not restart PC after install
618     : server = public IP address
619     : domain = LocalDomain always
620     : ALLUSERS=2 installs this for all users on the PC; case sensitive command.
621     #>
622 #>
623 ##############################################################################################
624                     ### APPLICATION INSTALLATION SECTION <END> ###
625 ##############################################################################################
626 #endregion
627
628 ##############################################################################################
629                     ### SYSTEM UPDATES SECTION <START> ###
630 ##############################################################################################
631 #region
632
633 ##DELL COMMAND SECTION START.
634
635 # Define the possible directory paths for Dell Command Update
636     $DCUdirPath1 = "C:\Program Files (x86)\Dell\CommandUpdate\"
637     $DCUdirPath2 = "C:\Program Files\Dell\CommandUpdate\"
638     $global:DCUdirPath = $null  # Initialize the unified directory path variable
639
640 # Function to check if either path exists and set $DCUdirPath
641 function Set-DCUPath {
642     if (Test-Path $DCUdirPath1) {
643         $global:DCUdirPath = $DCUdirPath1
644         Write-Output "Using Dell Command Update Path: $DCUdirPath1" -ForegroundColor
        Green
645     }
646     elseif (Test-Path $DCUdirPath2) {
647         $global:DCUdirPath = $DCUdirPath2
648         Write-Output "Using Dell Command Update Path: $DCUdirPath2" -ForegroundColor
        Green
649     }
650     else {
651         Write-Output "Dell Command Update is not installed on this system." -
        ForegroundColor Red
652         $global:DCUdirPath = $null
653     }
654 }
655
656 ## FUNCTION TO RUN DELL UPDATES
657 function Run-DellUpdates {
658     if ($null -ne $global:DCUdirPath) {
659         $DCUexePath = Join-Path -Path $global:DCUdirPath -ChildPath "dcu-cli.exe"
660         if (Test-Path $DCUexePath) {
661             try {
662                 # Run the scan
663                 Write-Output "Starting Dell Command Update scan..." -ForegroundColor Cyan
664                 $scanResult = & $DCUexePath /scan 2>&1
665                 Write-Output $scanResult -ForegroundColor Green
666
667                 # Apply updates with reboot DISabled ( PC will restart at the end of
                this script)
668                 Write-Output "Applying updates..." -ForegroundColor Cyan
669                 $applyResult = & $DCUexePath /applyUpdates -reboot=Disable 2>&1
670                 Write-Output $applyResult -ForegroundColor Cyan
671
672                 # Check for errors
673                 if ($scanResult -match "Error" -or $applyResult -match "Error") {
674                     Write-Output "An error occurred during the update process." -
                    ForegroundColor Red
675                 } else {
```

```
                    Write-Output "Dell updates completed successfully." -ForegroundColor Green
                }
            }
            catch {
                Write-Output "An unexpected error occurred with Dell Command Update: $_" -ForegroundColor Red
            }
        }
        else {
            Write-Output "Dell Command Update executable not found in $global:DCUdirPath" -ForegroundColor Red
        }
    }
    else {
        Write-Output "Dell Command Update path is not set." -ForegroundColor Red
    }
}

#### FUNCTION TO CHECK AND RUN DELL UPDATES ####
function Check-And-Run-DCU {
    Set-DCUPath  # Set the correct path
    Run-DellUpdates  # Run updates if the path is valid
}

#### CALL THE FUNCTION TO CHECK AND RUN DELL UPDATES ####
Check-And-Run-DCU

##### RUN WINDOWS UPDATE ####

# Enable PowerShell 7 to install Windows updates.

Import-Module PSWindowsUpdate

# Import Windows Update PS module (needs to have PS ver 7 installed)
try {
    $maxRetries = 2
    $retryCount = 0
    $success = $false

    while (-not $success -and $retryCount -lt $maxRetries) {
    try {
    # Install Windows Updates without AutoReboot
        Write-Host "Starting Windows Updates and will automatically reboot the machine when complete." -ForegroundColor Yellow
        Get-WindowsUpdate -AcceptAll -Install -ErrorAction Stop -AutoReboot:$false
    # If updates are installed successfully, set success flag to true
        $success = $true
        Write-Host "Installed Windows Updates Successfully."
        Log-Message "Installed Windows Updates Successfully."
    }
    catch { $retryCount++
        if ($retryCount -lt $maxRetries) {
         Write-Host "Windows Update Installation failed. Retrying attempt #$retryCount..." -ForegroundColor Yellow
        }
        else {
            Write-Host "Windows Update Installation failed after $retryCount attempts." -ForegroundColor Red
            Log-Message "Windows Update Installation failed after $retryCount attempts."
             }
          }
    }

# If after retries it fails, handle the failure logging and reboot
    if (-not $success) {
        Write-Host "Windows Update installation failed after $maxRetries attempts. Forcing a reboot in 5 seconds..." -ForegroundColor Red
        Log-Message "Windows Update installation failed after $maxRetries attempts.
```

```
            Forcing a reboot in 5 seconds..."
        } else {
            Write-Host "Updates installed. Rebooting in 5 seconds..." -ForegroundColor Green
        }
    }
    catch {
        # Log any errors during the update process
        Write-Host "ERROR: Failed to install Windows Updates. Error: $($_.Exception.Message)"
         -ForegroundColor Red
        Log-Message "ERROR: Failed to install Windows Updates. Error:
        $($_.Exception.Message)"

    }

    # End logging to setup log file.
    Log-Message "Initial Setup script execution ended."


    # Define the path to the PowerShell script you want to run
    $scriptPath = "C:\Sources\new_setup_part_2.ps1"

    # Create the scheduled task action to run PowerShell as administrator and execute the
    script
    $action = New-ScheduledTaskAction -Execute "powershell.exe" -Argument "-ExecutionPolicy
    Bypass -File $scriptPath"

    # Set the trigger to run at startup (on the next reboot)
    $trigger = New-ScheduledTaskTrigger -AtStartup

    # Define the task settings, e.g., run as highest privileges
    $settings = New-ScheduledTaskSettingsSet -AllowStartIfOnBatteries -
    DontStopIfGoingOnBatteries -StartWhenAvailable -AllowHardTerminate

    # Specify the task to run as SYSTEM (Administrator) and on the next reboot
    $principal = New-ScheduledTaskPrincipal -UserId "SYSTEM" -LogonType ServiceAccount

    # Register the scheduled task with the Task Scheduler
    Register-ScheduledTask -Action $action -Trigger $trigger -Settings $settings -Principal
    $principal -TaskName "RunPowerShellScriptAtReboot" -Description "Runs PowerShell script
    at the next reboot as administrator."


    # Force reboot after 5-second delay before reboot to allow logging to finalize.
    shutdown.exe /r /f /t 5

    ##############################################################################
    #endregion

                            <# end of script #>
```