I'm not a robot

PowerShell is an open source programming language designed for automation.PowerShell by Example, modeled after Go by Example, is a hands-on introduction to PowerShell using annotated example programs. Check out the first example or browse the full list below.To execute the scripts make sure you have at least PowerShell 5 installed.Homeby Sander Stad |source |licenseIll share 10 handy scripts Ive crafted to automate everyday tasks and boost your productivity. Whether youre a seasoned scripter or just just someone looking to enhance productivity with cool automation, these gems will make your life easier and save you from tedious clicks. These scripts cover a range of scenarios from system maintenance to data manipulation. Lets get started!Optimize Performance and Stay Informed with this Free Space Monitoring Script:Maintaining adequate free space on your C drive is crucial for computer performance and stability. To simplify this task, Ive developed a script that automatically checks the free space on your C drive daily, ensuring you stay informed about its availability. Set your own low-disk space threshold (e.g. 10GB free space) to trigger proactive alerts. You can schedule this to run once daily.# Define the threshold$threshold = 30*1GB# Get the C drive information$drive = Get-PSDrive -Name C# Calculate the free space in bytes$freeSpaceBytes = $drive.Free# Convert the free space to GB$freeSpaceGB = [Math]::Round($freeSpaceBytes / 1GB, 2)# Compare the free space with the thresholdif ($freeSpaceBytes -lt $threshold) { Write-Host "Warning: Free space on C drive is below 30GB. Current free space is $freeSpaceGB GB."} else { Write-Host "The free space on C drive is sufficient. Current free space is $freeSpaceGB GB."}2. Unveiling App Crashes: A PowerShell Script for Application Data InspectionTarget any application: Specify the application name and the script delves into its crash data, Gather vital information like crash event time, crash type, and even potential error messages.# Prompt the user for ProviderName input$providerName = Read-Host -Prompt 'Enter the ProviderName'# Define the filter criteria with user input$filterHashtable = @{ LogName = 'Application' ProviderName = $providerName}# Retrieve the latest event that matches the filter criteria$lastEvent = Get-WinEvent -FilterHashtable $filterHashtable -MaxEvents 1# Print message if an event is foundif ($lastEvent) { $lastEvent.Message} else { Write-Host "No events found for ProviderName: $providerName"}3. Find Out Where in the World an IP LivesSeen movie scenes where high tech guys tracking someone using IP? You can do that simply two line of script with help of ip-api.coms apitry { $IPaddress = Read-Host "Enter IP address to locate" $result = Invoke-RestMethod -Method Get -Uri " IPaddress" Write-Output $result} catch { " Error in line $($_.InvocationInfo.ScriptLineNumber): $($Error[0])" exit 1}4. Secure Your PDFs Fortress with This PowerShell ScriptProtect your sensitive PDF documents with ease using this script that effortlessly applies password protection to multiple files within a folder. Efficiently secure large numbers of PDFs in a single operation, saving you time and effort. For this to work first install qpdf by choco install qpdf# Prompt the user for PDF folder and password input$pdfFolder = Read-Host -Prompt 'Enter the path to the folder containing PDF files'$password = Read-Host -Prompt 'Enter the password to be applied to the PDF files'# Get all PDF files in the folder$pdfFiles = Get-ChildItem -Path $pdfFolder -Filter *.pdf# Loop through each PDF file and password-protect itforeach ($pdfFile in $pdfFiles) { # Define the output file with the same name as the original file $outputFile = $pdfFile.FullName # Temporary filename for the original file $tempFile = "$($pdfFile.FullName).temp" # Rename the original file Rename-Item -Path $pdfFile.FullName -NewName $tempFile # Password-protect the PDF file using QPDF & qpdf --encrypt $password $password 256 -- "$tempFile" "$outputFile" # Remove the temporary file Remove-Item -Path $tempFile -Force Write-Host "Password protection completed for: $($pdfFile.FullName)"}Write-Host "Password protection completed for all PDF files in $pdfFolder."5. Declutter your desktop with This Recycle Bin Auto-CleanerTired of overflowing recycle bins cluttering your precious storage space? This PowerShell script offers a tidy solution, automatically emptying the bin when it exceeds a set threshold, keeping your system clean and efficienttry { $recycleBinPath = [System.IO.Path]::Combine($env:SystemRoot, 'Recycle Bin') $recycleBinSize = (Get-ChildItem $recycleBinPath -Recurse | Measure-Object -Property Length -Sum).Sum / 1GB if ($recycleBinSize -gt 30) { #change threshold as needed Clear-RecycleBin -Confirm:$false if ($lastExitCode -ne "0") { throw "'Clear-RecycleBin' failed" } } else { Write-Output "Recycle bin size is less than or equal to 30GB. No action taken." exit 0 # success } } catch { " Error in line $($_.InvocationInfo.ScriptLineNumber): $($Error[0])" exit 1}6. Zip it Up and Ship It Out! Compress Multiple FoldersThis script that effortlessly zips up multiple folders within a specified directory saving you time and tedious manual actions and streamlining your file management and saving precious storage space or just for sharing purposes.# Prompt the user for source and destination folders$SourceFolder = Read-Host "Enter the source folder"$DestinationFolder = Read-Host "Enter the destination folder for zipped files"# Check if the source folder existsif (-Not (Test-Path $SourceFolder -PathType Container)) { Throw "The source directory $SourceFolder does not exist, please specify an existing directory"}# Check if the destination folder exists, if not, create itif (-Not (Test-Path $DestinationFolder -PathType Container)) { New-Item -ItemType Directory -Path $DestinationFolder | Out-Null}$date = Get-Date -format "yyyy-MM-dd"$folders = Get-ChildItem -Path $SourceFolder -Directoryforeach ($folder in $folders) { $dirPath = $folder.FullName $destinationPath = Join-Path $DestinationFolder "$($folder.Name)_$date.zip" Compress-Archive -Path $dirPath -CompressionLevel 'Fastest' -DestinationPath $destinationPath}Write-Host "Compressed!!"7. Save time on Unzipping headacheTired of manually extracting files from multiple zip archives? This script automates the process, liberating your data with a few simple commands. For each zip file found, the script will expertly extract its contents, releasing your files from their compressed confines.# Ask the user for the source directory containing the zip files$sourceDirectory = Read-Host "Please enter the source folder path"# Ask the user for the destination directory where the unzipped files will be placed$destinationDirectory = Read-Host "Please enter the destination folder path"# Ensure the destination directory existsif (-not (Test-Path -Path $destinationDirectory)) { New-Item -ItemType Directory -Path $destinationDirectory | Out-Null}# Get all zip files from the source directory$zipFiles = Get-ChildItem -Path $sourceDirectory -Filter *.zip -Recurse# Loop through each zip file and extract it to the destination directoryforeach ($zipFile in $zipFiles) { # Create a subfolder in the destination directory with the same name as the zip file (without extension) $subfolder = Join-Path -Path $destinationDirectory -ChildPath $zipFile.BaseName New-Item -ItemType Directory -Path $subfolder -Force | Out-Null # Unzip the file to the subfolder Expand-Archive -LiteralPath $zipFile.FullName -DestinationPath $subfolder -Force}Write-Host "All files have been unzipped to $destinationDirectory"8. Add Some Daily Surprise to Your Desktop with This Wallpaper-Shuffling ScriptKeep your desktop fresh and inspiring with this PowerShell script that automatically picks a random image from your favorite folder and sets it as your wallpaper. schedule this to run once everyday and get a daily dose of visual delight!# Get the screen dimensionsAdd-Type -AssemblyName System.Windows.Forms$screen = [System.Windows.Forms.Screen]::PrimaryScreen$screenWidth = $screen.Bounds.Width$screenHeight = $screen.Bounds.Height# Add the C# code to define the SystemParametersInfo functionAdd-Type -TypeDefinition @"using System;using System.Runtime.InteropServices;public class WallpaperChanger { [DllImport("user32.dll", CharSet = CharSet.Auto)] public static extern int SystemParametersInfo(int uAction, int uParam, string lpvParam, int fuWinIni);}"@# Add references to the assemblies that contain the Screen and Graphics classesAdd-Type -AssemblyName System.Windows.FormsAdd-Type -AssemblyName System.Drawing# Set the path to the folder containing images$imageFolderPath = "C:\Users\UserName\Pictures\wallpapers" #change to your custom path with your images# Get all image files in the folder$imageFiles = Get-ChildItem -Path $imageFolderPath -Include *.jpg,*.jpeg,*.png,*.bmp,*.gif -Recurse | Where-Object { !$_.PSIsContainer }# Check if there are any image filesif ($imageFiles.Count -gt 0) { # Randomly choose an image $randomImage = Get-Random -InputObject $imageFiles # Get the screen resolution $screen = [System.Windows.Forms.Screen]::PrimaryScreen $screenWidth = $screen.Bounds.Width $screenHeight = $screen.Bounds.Height # Load the chosen image $image = [System.Drawing.Image]::FromFile($randomImage.FullName) # Create a new Bitmap object with the screen resolution $resizedImage = New-Object Drawing.Bitmap $screenWidth, $screenHeight # Create a graphics object from the bitmap $graphics = [System.Drawing.Graphics]::FromImage($resizedImage) # Draw the image onto the bitmap, scaled to fit the screen resolution $graphics.DrawImage($image, 0, 0, $screenWidth, $screenHeight) # Save the bitmap as a temporary image file $tempImagePath = Join-Path ([IO.Path]::GetTempPath()) "wallpaper.bmp" $resizedImage.Save($tempImagePath, [System.Drawing.Imaging.ImageFormat]::Bmp) # Set the temporary image as the desktop wallpaper [WallpaperChanger]::SystemParametersInfo(20, 0, $tempImagePath, 3) Write-Host "Desktop wallpaper set to: $($randomImage.FullName)" # Dispose the graphics object and the image objects $graphics.Dispose() $image.Dispose() $resizedImage.Dispose()} else { Write-Host "No image files found in the specified folder."}9. Harness the Power of Data with This API-to-File ScriptHarness the Power of Data with This API-to-File ScriptUnlock the wealth of information residing in APIs and bring it offline for seamless analysis and integration with this versatile PowerShell script. It effortlessly captures JSON responses from APIs and stores them both as JSON files for future use and as Excel files for convenient analysis and manipulation.# Import the moduleImport-Module ImportExcel# Ask the user for the API endpoint$apiEndpoint = Read-Host "Enter the API endpoint"# Make the GET request$response = Invoke-RestMethod $apiEndpoint# Ask the user for the export path of the Excel file$exportPath = Read-Host "Enter the path to export the Excel file"# Export the data to an Excel file$response | Export-Excel -Path $exportPath -AutoSize -Show# Ask the user for the export path of the JSON file$jsonExportPath = Read-Host "Enter the path to export the JSON file"# Save the data as a JSON file$response | ConvertTo-Json -Depth 100 | Out-File $jsonExportPathWrite-Host "Data exported to $exportPath (Excel) and $jsonExportPath (JSON)."10. Hush the Boombox, Save Your Eardrums: the Gentle Volume Minder!This script is not just a code snippet; its a shield against unexpected decibel disasters, a guardian of your sonic serenity. So unleash your inner DJ with confidence, knowing your audio adventures are always under the watchful eye of your automated volume tamer!Add-Type -TypeDefinition @"using System.Runtime.InteropServices;[Guid("5CDF2C82-841E-4546-9722-0CF74078229A"), InterfaceType(ComInterfaceType.InterfaceIsIUnknown)]interface IAudioEndpointVolume{ int f(); int g(); int h(); int i(); int SetMasterVolumeLevelScalar(float fLevel, System.Guid pguidEventContext); int j(); int GetMasterVolumeLevelScalar(out float pfLevel); int k(); int l(); int m(); int n(); int SetMute([MarshalAs(UnmanagedType.Bool)] bool bMute, System.Guid pguidEventContext); int GetMute(out bool pbMute);}[Guid("D666063F-1587-4E43-81F1-B948E807363F"), InterfaceType(ComInterfaceType.InterfaceIsIUnknown)]interface IMMDevice{ int Activate(ref System.Guid id, int clsCtx, int activationParams, out IAudioEndpointVolume aev);}[Guid("A95664D2-9614-4F35-A746-DE8DB63617E6"), InterfaceType(ComInterfaceType.InterfaceIsIUnknown)]interface IMMDeviceEnumerator{ int f(); int GetDefaultAudioEndpoint(int dataFlow, int role, out IMMDevice endpoint);}[ComImport, Guid("BCDE0395-E52F-467C-8E3D-C4579291692E")] class MMDeviceEnumeratorComObject { }public class Audio{ static IAudioEndpointVolume Vol() { var enumerator = new MMDeviceEnumeratorComObject() as IMMDeviceEnumerator; IMMDevice dev = null; Marshal.ThrowExceptionForHR(enumerator.GetDefaultAudioEndpoint(/*eRender*/ 0, /*eMultimedia*/ 1, out dev)); IAudioEndpointVolume epv = null; var epvid = typeof(IAudioEndpointVolume).GUID; Marshal.ThrowExceptionForHR(dev.Activate(ref epvid, /*CLSCTX_ALL*/ 23, 0, out epv)); return epv; } public static float Volume { get { float v = -1; Marshal.ThrowExceptionForHR(Vol().GetMasterVolumeLevelScalar(out v)); return v; } set { Marshal.ThrowExceptionForHR(Vol().SetMasterVolumeLevelScalar(value, System.Guid.Empty)); } } public static bool Mute { get { bool mute; Marshal.ThrowExceptionForHR(Vol().GetMute(out mute)); return mute; } set { Marshal.ThrowExceptionForHR(Vol().SetMute(value, System.Guid.Empty)); } }}"@function Set-Volume([Parameter(Mandatory=$true)][ValidateRange(0,100)][Int]$percent) { try { # Create the Windows Shell object. $obj = New-Object -ComObject WScript.Shell # Lower the volume if it's higher than the specified level. do { $obj.SendKeys([char]174) # Volume down } while ([Audio]::Volume -gt $percent/100) # Raise the volume if it's lower than the specified level. do { $obj.SendKeys([char]175) # Volume up } while ([Audio]::Volume -lt $percent/100) } catch { " Error in line $($_.InvocationInfo.ScriptLineNumber): $($Error[0])" }}$checkIntervalMinutes = 10 #set this to your desired interval$highVolumeCount = 0while ($true) { Start-Sleep -Seconds ($checkIntervalMinutes * 60) # Convert minutes to seconds $currentVolume = [Audio]::Volume if ($currentVolume -gt 0.7) { $highVolumeCount++ if ($highVolumeCount -ge 1) { # Adjusted to 1 for 10 minutes interval Set-Volume -percent 60 #Set this to your desired volume to reduce $highVolumeCount = 0 } } else { $highVolumeCount = 0 }}HAPPY AUTOMATION!Skip to contentJordan Hammond|Updated November 20, 2024System administrators love a good shortcut, especially when it comes to alleviating the aggravation that comes with managing computers, solving user issues, and putting out fires around the office. Enter PowerShell.Though theres a wide variety of great scripting languages out there, it doesn't get much better than the PowerShell scripting language. And now, it can run across platforms, making it a useful tool no matter the environment you find yourself working in. Sysadmins have too much on their plates, so task automation is becoming mandatory. No better place to start learning PowerShell starting from the basics.PowerShell saves scripts in the .ps1 format. Feel free to use your own custom folder and file names. For our demonstration, we created both a file and a folder:C:\Scripts\My First Script.ps1Create the new PowerShell script file, and add the Write-Host cmdlet (cmdlet is another word for a PowerShell command).Write-Host "Hello, World!"Save your .ps1 script file, and return to the PowerShell window. When running PowerShell scripts, the most common method is to call it in the PowerShell terminal. (You can also use the PowerShell ISE or VS CodeVS Code.)& "C:\Scripts\My First Script.ps1"Go ahead and try that command. You should get an error that says scripts are disabled on your system. This is for security reasons.In order to prevent malicious scripts from running on your system, PowerShell enforces an execution policy. To use our newly created script, we have to modify our execution policy to allow our PowerShell example script to run. There are four execution policies:Since we have not digitally signed our new PowerShell example script, our options for the execution policy are limited to RemoteSigned and Unrestricted. We are going to change it to RemoteSigned.To change the execution policy, open PowerShell as an administrator (the command fails otherwise), and run the following command:Set-ExecutionPolicy RemoteSignedThe Set-ExecutionPolicy cmdlet asks to verify that you really want to change the script execution policy. Go ahead and select Y for yes, then close and reopen your PowerShell window.After restarting the PowerShell window, try running your .ps1 script again.& "C:\Scripts\My First Script.ps1"It should write back, "Hello, World!" to the Windows PowerShell console:Congratulations you just wrote your first PowerShell script! Easily run PowerShell scripts on remote devicesNeed to run your awesome PowerShell scripts on remote devices? PDQ Connect can easily execute PowerShell scripts on any managed device with an active internet connection.When looking for ideas on what you can do with beginner scripts, just think about things you need to do manually and see if can grab the information. One good quick win is checking the ACL of file share:Get-Acl "\\fileshare\folder" | Select-Object -ExpandProperty AccessOr try clearing out a temp folder that is taking up space with unneeded documents:Get-ChildItem C:\temp | Remove-ItemOr test if a registry key is on a machine: Get-ItemProperty -Path "HKLM:\SOFTWARE\Admin Arsenal\PDQ Deploy"Those are all quick wins that you can grab out of the box. The entire world opens once you dive into PowerShell modules and start with scripts against critical systems, like Azure, VMWare, AWS, or Active Directory.Learn PowerShell with the prosEven the most experienced PowerShell aficionados are constantly learning. Ready to dive into more advanced functions? Check out The PowerShell Podcast to learn more about key people, resources, and modules. You now have the amazing power to create and run your own scripts and cmdlets. Jordan HammondJordan had spent his life wondering why tasks he didnt like to do had no options to complete themselves. Eventually he had to make that happen on his own. It turned out that he enjoyed making tasks complete themselves, and PDQ thought that is something he should talk about on the internet. Automating daily tasks, extracting information from systems, or managing Microsoft 365? PowerShell scripts really make your daily work a lot easier. When you are working in IT then you cant get around PowerShell. PowerShell is an advanced command line interface (CLI) and scripting language that can be used on Windows, Linux, and macOS. With the help of cmdlets, we can perform tasks like retrieving users from the Active Directory or testing the network connection of a server. We can combine these tasks and processes into scripts that we can quickly run on multiple computers or schedule as a daily task. In this article, I will explain how you can create your own PowerShell Scripts. Which tools you can use, how to format your scripts and some general tips to get started. At the end of the article, you will also find a template you can use for your scripts. Most people that start writing PowerShell scripts use a simple notepad tool, like Notepad++. It works for small scripts, but a good editor makes writing PowerShell scripts much easier. They come with syntax highlighting, autocomplete functions, error detection, etc. And what I like the most is that you can create a project, allowing you to quickly switch between files, and keep your files organized. For PowerShell, one of the best free editors to start with is Visual Studio Code. This editor, from Microsoft, is completely free and can be used on Windows, Linux, and macOS. To use the editor with PowerShell, you will need to install a plugin (extension). First, install the Visual Studio Code using the installer. Simply click next on all screens. Click on Extensions Search for PowerShell Install the PowerShell extension from Microsoft PowerShell Editor Visual Studio Code To create our first PowerShell script we will need to create a new file. To keep organized, we will create a new folder scripts where we are going to store our files. In Visual Studio Code: Open the Explorer Choose Open Folder Create a new folder, scripts, in your OneDrive for example Click Select Folder You will get a prompt if you trust the files in this folder, make sure you check Trust the authors of all files in the parent folder and click Yes, I Trust the authors. To create a new file, right-click in the editor (or click on the New file icon) and create your first PowerShell script. You can also create a subfolder to keep your scripts organized of course. Before we start writing our PowerShell script, its good to know how you can run or test your scripts. One of the common issues, when you try to run your script, is the error Running scripts is disabled on this system. To solve this we will have to change the execution permission on your Windows 10 or 11 computer. Open PowerShell and type the command below to change the execution policy. Read more about the PowerShell execution policy in this article. Set-ExecutionPolicy -ExecutionPolicy RemoteSigned# Verify the setting withGet-ExecutionPolicyWith the execution policy set, we can run our scripts without any permissions errors. Now there are multiple options to run a PowerShell script, read more about them in this article, but for this guide, we will focus on the following two: Using the built-in terminal in Visual Studio Code Run the script in PowerShell One of the advantages when using Visual Studio Code as an editor for your PowerShell scripts is that you can test and run your scripts in the built-in terminal. You can run the whole script by pressing F5 or run a selection of lines by pressing F8. Running PowerShell code Another method is using PowerShell itself to run your script. Open Windows PowerShell or Windows Terminal (right-click on Start) and navigate to the folder where your script is saved. Type the filename of your script and press enter. Using Windows Terminal In the command prompt, we have commands, which are built-in functions (commands) that perform a task. For example, if you type hostname in the command prompt, it will show your computer name. Or the command dir will list the contents of the current directory. Command Prompt commandsIn PowerShell, we have command-lets (cmdlets). You can recognize a cmdlet by its naming because it exists of a Verb-Noun pair. This naming convention helps you to understand what a particular cmdlet does. NoteIn some examples below I am using cmdlets from the Active Directory module. Make sure that you have the module installed if you want to follow the steps. Run the following command to install the module:Add-WindowsCapability online Name Rsat.ActiveDirectory.DS-LDS.Tools~~~~0.0.1.0 For example, the Get-ComputerInfo cmdlet gets all the computer information from your computer. To show the contents of a folder, we can use Get-ChildItem, for example. Besides the built-in cmdlets, you can also install modules in PowerShell. These modules can, for example, be used to work with Exchange Online or Azure AD. Each module comes with its own cmdlets for its specific task. PowerShell Cheat SheetMake sure that you also checkout and download the PowerShell Cheat Sheet The are a lot of approved verbs that you can use when you are creating your own PowerShell commands or functions. But the most common verbs that you will come across are: VerbActionExampleAddAdds or appends are resources to another itemAdd-MailboxPermissionClearRemoves all resources from a container, but doesnt delete the containerClear-HostConnectMakes a connection to another systemConnect-AzureADDisconnectBreak the connection with the other systemDisconnect-AzureADGetRetrieves data from a resourceGet-ChildItemNewCreates a new resourceNew-MailboxRemoveRemove a resource from a containerRemove-MailboxSetReplaces data in an existing resourceSet-MailboxSelectSelects a resource in a containerSelect-ObjectPowerShell Verbs When writing scripts you will often need to store data temporarily, so you can use it later in your script. For this we use variables. In PowerShell we dont need to initialize the variables, we can just create them when needed. Variables can not only store data, like strings, and integers, but also the complete output of cmdlets. Lets start with a simple example, we take the Get-Computer cmdlet from before. The cmdlet returns your computer name: NoteThe contents after the # in a PowerShell script is a comment. Comments are used to explain what a functions does and to make your code more reable. hostname# ResultLazyBookWe dont want to just show the computer name, but instead, we want to include the computer name inside a nice string. In this case, we can first store the result of the Get-Computername cmdlet into a variable, which we will call computername. And then include this variable inside the string: # Store computer name inside the variable$computername = hostname# Include the variable inside the stringWrite-Host "The name of your computer is $computername"# ResultThe name of your computer is LazyBookIn PowerShell, we can use comparison operators to compare or find matching values. By default, the operators are case-insensitive, but you can pace a c before an operator to make it case-sensitive. For example: 'lazyadmin' -eq 'LazyAdmin'# ResultTrue'lazyadmin' -ceq 'LazyAdmin'# ResultFalseThe operators that we can use in PowerShell are: OperatorCounter-Part operatorDescription-eq-neEqual or not equal-gt-ltGreater or less than-geGreat than or equal to-leLess than or equal to-Like-NotLikeMatch a string using * wildcard or not-Match-NotMatchMatches or not the specified regular expression-Contains-NotContainsCollection contains a specified value or not-In-NotInSpecified value in collection or not-ReplaceReplace specified valuePowerShell Operators We can use these operators in combination with for example If-Else statements. If statements allow us to check if a particular comparison is true or not. Depending on the outcome we can execute a piece of code or skip to another part. Lets take the computer name example again. We have stored the computer name inside the variable. Now lets check if the computers name is my laptop: $computername = hostname# Check if the computer name equal LazyBookif ($computername -eq 'LazyBook') { Write-Host "This computer is from LazyAdmin"}else{ Write-host "This is someone else's computer"}Our computer names start with LT for laptops and PC for desktops. So we could determine if the device is a laptop or not based on the computer name. For this, we are going to use an if statement with an -like operator. The like operator accepts a wildcard, so we can, for example, check if a string starts with LT in this case $computername = hostname# Or better is$computername = $env:COMPUTERNAME# Check if the computer name start with Lazyif ($computername -like 'Lazy*') { Write-Host "This is a laptop"}else{ Write-host "This is something else"} Besides variables, there is another way to pass data through to another cmdlet in PowerShell, which is using the pipeline operator | . The pipeline operator passes the results of the command to the next command. The most common example of piping is formating or selecting the result of a cmdlet. Lets take the following example, the cmdlet Get-NetIPAddress returns the IP Address configuration of all your network interfaces. By default, it will return the results in a list format Get-NetIPAddress To make the results more readable we can format the results into a tablet or select only the properties that we need. We do this by piping the cmdlet format-table (ft) behind it or the cmdlet select-object (select): Get-NetIPAddress | FT# Or select the fieldsGet-NetIPAddress | Select InterfaceAlias, IPAddress, PrefixOrigin Now, this is a simple example of piping cmdlets. But lets take a look at a more advanced use of piping cmdlets. We are going to collect all user mailboxes, from each mailbox we are going to look up the mailbox statistics, select only the fields that we need, and export the results to a CSV file. Without the pipe operator, we would need to write a code similar to this: NoteGet-EXOMailbox cmdlet is part of the Exchange Online module. Make sure that you have installed it if you want to follow the steps below. $mailboxes = Get-EXOMailbox -RecipientTypeDetails UserMailbox$mailboxes.ForEach({ $Mailboxstats = Get-EXOMailboxStatistics -Identity $_.Identity $fields = Select-Object -InputObject $Mailboxstats -Property DisplayName,ItemCount,TotalItemSize Export-CSV -InputObject $fields -Path c:\temp\file.csv -Append})But with piping in PowerShell we can simply do the following: Get-EXOMailbox -RecipientTypeDetails UserMailbox | Get-EXOMailboxStatistics | Select DisplayName, ItemCount, TotalItemSize | Export-CSV c:\temp\filename.csv Arrays and hashtables can be used to store a collection of data. Hashtables use a key value principle where you need to define the key before you can store the value. Arrays use an automatically generated index to store the values. To create an array we can simply assign multiple values to a variable, separating each of them with a common. For example: # Create an array of fruits$array = 'apple','raspberry','kiwi' Another option is to first initialize the array and add values to the array later on. To create an empty array we will use the @ symbol followed by parentheses: # Create an empty array$fruits = @()# Add content to the array$fruits += "apple" Hashtables are also known as associative arrays, they are used when you need t store data in a more structured manner. Instead of a numbered index, its based on a key-value pair, where you need to define the key yourself. To create an empty hashtable you will need to use curly brackets and the @ symbol: # Create empty hashtable$hashTable = @{}You could for example use a hashtable to store the server IP Addresses $serverIps= @{ 'la-srv-lab02' = '192.168.10.2' 'la-srv-db01' = '192.168.10.100'}# ResultName Value---- -----la-srv-lab02 192.168.10.2la-srv-db01 192.168.10.100Or as a config file inside your script: $mail = @{ SmtpServer = 'smtp.contoso.com' To = 'johndoe@lazyadmin.nl' From = 'info@contoso.com' Subject = 'super long subject goes here' Body = 'Test email from PowerShell' Priority = High} Looping through data is one of the common tasks in any scripting language. ForEach loops allow you to go through each item in a collection and do something with that item. For example, we take the array of fruits and write each item (fruit) to the console: $fruits = @('apple','pear','banana','lemon','lime','mango')# Foreach blockForeach ($fruit in $fruits) { Write-Host $fruit;}# Shorthand$fruits.foreach({ Write-Host $_;}) In the example above we only used a simple array, but you can also use ForEach on objects. Inside the ForEach block you can access each property of the object, for example, if we get all the mailboxes, we can access the display name as follows: $mailboxes = Get-EXOMailbox -RecipientTypeDetails UserMailbox$mailboxes.ForEach({ # Write the displayname of each mailbox Write-host $_.DisplayName}) Besides ForEach loops, we can also use While and Do-While loops. A while loop will only run when a condition is met, the Do-While always runs once and as long as the condition is true. # Do While loopDo { Write-Host "Online" Start-Sleep 5}While (Test-Connection -ComputerName 8.8.8.8 -Quiet -Count 1)Write-Host "Offline"# While loop$i = 0;$path = "C:\temp"While ($i -lt 10) { # Do Something $newFile = "$path\while_test_file_" + $i + ".txt"; New-Item $newFile $i++;} Learn more about For loops, ForEach statements, and Do While loops in this article. Try-catch blocks are used to handle errors in a proper way. Normally when a function doesnt work or runs into an error, the script will simply stop and throw an error. Sometimes this is fine, but on other occasions, you might want to show a more readable error or simply continue. For example, when you are updating multiple users in the Active Directory using a ForEach loop. When one of the user accounts doesnt exist, the script will run into an error and stop. But a better solution would be if the script outputs the name of the users it didnt update and just continues with the next one. This is where Try-Catch blocks come in. Taking the example above, the following code block will try to find and update the Azure AD user, if an error occurs in the Try block, then the Catch part will show an error. $users.ForEach{ Try{ # Find the user to update $ADUser = Get-AzureAdUser -SearchString $_.name # Update the job title Set-AzureAdUser -ObjectId $ADUser.ObjectId -JobTitle $_.jobtitle } Catch{ Write-Host ("Failed to update " + $($_.name)) -ForegroundColor Red }} Now, this is a basic implementation of a try-catch block, its even possible to use multiple catch blocks on a single Try statement, allowing you to catch different errors. Read more about Try-Catch blocks in this in-depth article. You should now have a brief understanding of what tools you can use in PowerShell to create scripts. So lets take a look at how we combine this into true PowerShell scripts. For the examples below we are going to create a small script that creates test files in a given folder. Below you will find the basic principle of the script. We have a path, hardcoded in the script, an array with the numbers 1 to 10, and a ForEach loop. With the examples below we are going to enhance this script to a true PowerShell script. $path = "C:\temp"1..10 | ForEach-Object { $newFile = "$path\test_file_$_.txt"; New-Item $newFile} When creating PowerShell scripts its always a good idea to add documentation to your script. The documentation is placed at the top of your script in a comment block and describes what the script does, a couple of examples on how to use the script, and a notes block with the author, version, date, etc. So for our test file script, we can add the following description at the beginning of our file: