

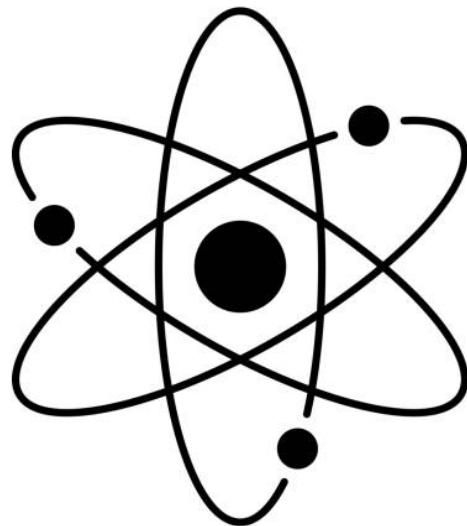
# APPLIED ATOMICS

---

DATA STRUCTURES • ALGORITHMS • MORE

PRACTICAL SCRIPTING CONCEPTS

INTERMEDIATE LEVEL



EDITION



EDDIE JACKSON



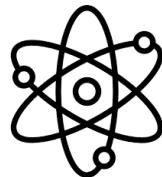
---

# Applied Atomics

Data Structures • Algorithms • More

---

## Practical Scripting Concepts



AN ATOMIC STRUCTURE SERIES

FIRST EDITION

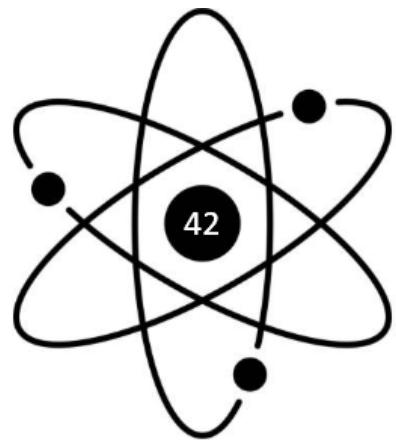
**Eddie Jackson**  
( MrNetTek )

COMMLINK PUBLISHING

2025



*Dedicated to Kaplan,  
for fostering a culture of innovation, collaboration, and dedication. Your commitment to  
excellence and your unwavering support have been instrumental in my growth as a  
professional. May we continue to push boundaries, learn, and grow together.*



**"I didn't study programming. I studied cause and effect."**

**Eddie S. Jackson**



# Applied Atomics

## Data Structures • Algorithms • More

---

PRINCIPLES & PATTERNS IN VARIOUS SCRIPTING AND PROGRAMMING LANGUAGES

### Table of Contents

1. Greetings, Fellow Explorer!.....	20
2. Introduction.....	25
3. Definitions.....	27
4. Important Links.....	35
Books.....	35
Tools.....	36
Websites.....	36

### SCRIPTING LANGUAGES

5. Batch Script.....	37
Batch: Design.....	40
Basic Construction.....	40
Batch: Enable Delayed Expansion.....	41
Batch: String Operations.....	42
Remove Leading Space.....	42
Remove Trailing Space.....	42
Detect & Remove Leading Space.....	43
Replace Underscore with Hyphen.....	44
Create Formatted Timestamp.....	44



<b>Batch: User Operations.....</b>	<b>46</b>
Return User SID.....	46
Method 1.....	46
Method 2.....	47
Method 3.....	47
Method 4.....	48
Method 5.....	49
Return Current User.....	50
Return Current User Domain.....	50
Return UPN.....	51
Method 1.....	51
Method 2.....	52
Method 3.....	53
Return User Profile Path.....	54
Detect if User is 'Elevated' as Admin.....	55
Add User to Administrators Group.....	55
Return User Session Information.....	56
Return User SIDs from HKEY_USERS.....	59
Return Username, SID, User Profile & More.....	60
<b>Batch: File &amp; Folder Operations.....</b>	<b>64</b>
Copy File.....	64
Xcopy, Files & Folders.....	65
Robocopy, Files & Folders.....	65
Copy File to Folder.....	65
Copy Folder to Folder, Keep Source.....	66
Copy Folder to Folder, Delete Source.....	67
Delete File.....	67
Delete Folder.....	68
Set Current Working Directory.....	68
Return File LastWriteTime.....	69
Return File CreationTime.....	69
Copy File to Each User Profile.....	71
Delete File from Each User Profile.....	72
Read Contents of File.....	73
Check if Folder is Empty.....	74
Delete Folders Older Than 30 Days.....	75



Return MD5 for File.....	76
Verify MD5 for File.....	76
Return MD5s for Temp Folder Files.....	77
Set Permissions on Files or Folders.....	78
Take Ownership of Files or Folders.....	78
Unblock Downloaded File.....	79
<b>Batch: Text File Operations.....</b>	<b>80</b>
Create Text File.....	80
Write to Text File.....	80
Append to Text File.....	81
Read from Text File.....	81
Read from Text File, Line by Line.....	82
Read from Text File, Type.exe.....	83
Search & Replace String in Text File.....	84
Sort Text File Ascending/Descending.....	85
Merge Two Text Files.....	87
Merge Two Text Files, Remove Dups.....	89
Return Specific Line in Text File.....	91
<b>Batch: Process Operations.....</b>	<b>93</b>
Start Process.....	93
Kill Process.....	93
Detect Running Process & Wait.....	93
Detect Running Process & Proceed.....	94
Wait for Multiple Processes to End.....	95
Capture and Use PID to Terminate.....	96
<b>Batch: Registry Operations.....</b>	<b>97</b>
Add Reg Key.....	97
Delete Reg Key.....	97
Delete Reg Key Value.....	98
Read Reg Key Value.....	98
Detect Reg Key Value.....	99
Detect Reg Key Value, Return True or False.....	99
Detect Reg Key Data, Return True or False.....	100
Scan Reg for String.....	100
Return Reg Subkeys Based on Data.....	101
Add Reg Key for All Users.....	102



Delete Reg Key for All Users.....	102
Detect Reg Key, GOTO Label.....	103
<b>Batch: Logical Constructs.....</b>	<b>105</b>
IF Condition, Control Flow.....	105
IF-ELSE Condition, Control Flow.....	106
FOR Loop, Various Implementations.....	107
GOTO, Decision-making.....	108
CALL, Decision-making.....	109
FOR Loop with GOTO.....	110
IF Condition with CALL.....	111
Simulate a WHILE LOOP.....	112
<b>Batch: Data Structures.....</b>	<b>113</b>
Array Simulation.....	113
Stack Simulation.....	115
Queue Simulation.....	117
Hash Table Simulation.....	119
<b>Batch: Miscellaneous.....</b>	<b>121</b>
Create a Counter, 1-10.....	121
Create a Countdown Timer, 10-1.....	121
Detect FSLogix.....	121
Hide Output of Command.....	122
Set PowerShell Remote Signed.....	123
Launch PowerShell Script.ps1.....	123
Return OS Major Version.....	123
Detect if Current Computer is Online.....	124
Detect if Computers are Online.....	125
VDI Engine, Cycle VM List.....	126
Clear All Windows Logs.....	128
Import Certificate.....	128
Sign & Verify Executable with Certificate.....	129
Add Application to Windows Path.....	129
Create Network Share, Shared Folder.....	130
Animation (Windows Terminal).....	131
<b>Batch: Task Scheduler.....</b>	<b>132</b>
On Demand App Install.....	132
On Demand App Repair.....	133



On Demand App Download.....	133
Restart Print Spooler Every 24 Hours.....	134
Schedule PowerShell Script to Run.....	134
Schedule PowerShell Command to Run.....	135
Back up Documents Folder Weekly.....	135
Disk Cleanup 1.....	138
Disk Cleanup 2.....	138
Monitor High CPU Usage.....	140
<b>6. PowerShell.....</b>	<b>141</b>
<b>PowerShell: Design.....</b>	<b>144</b>
Basic Construction.....	144
<b>PowerShell: String Operations.....</b>	<b>146</b>
Remove Leading Space.....	146
Remove Trailing Space.....	146
Remove Leading & Trailing Space.....	146
Detect & Remove Leading Space.....	146
Replace Underscore with Hyphen.....	147
Check Length of String.....	147
Detect Pattern in String.....	148
Check if String is Empty or Null.....	149
Create Formatted Timestamp.....	149
<b>PowerShell: User Operations.....</b>	<b>150</b>
Return UPN.....	150
Method 1.....	150
Method 2.....	151
Method 3.....	152
Method 4.....	153
Return Current Username.....	155
Method 1.....	155
Method 2.....	155
Method 3.....	156
Method 4.....	157
Method 5.....	158
Method 6.....	159
Method 7.....	161



Return User SIDs.....	161
Method 1.....	161
Method 2.....	162
Method 3.....	163
Method 4.....	164
Method 5.....	164
Method 6.....	165
Return User Domain.....	167
Method 1.....	167
Method 2.....	167
Return User Profile Path.....	168
Method 1.....	168
Method 2.....	169
Return Appdata Folder.....	169
Return LocalAppData Folder.....	170
Add User to Group.....	171
Return Domain Users in the Administrators Group.....	173
Detect if User is in Administrators Group.....	173
Method1.....	173
Method 2.....	174
Detect if User is 'Elevated' as Admin.....	176
Method 1.....	176
Method 2.....	176
Return Usernames in Loop.....	177
Method 1.....	177
Method 2.....	178
Method 3.....	178
Return Usernames, SIDs, LocalAppData.....	179
<b>PowerShell: File &amp; Folder Operations.....</b>	<b>183</b>
Copy File.....	183
Copy File, If Like.....	183
Copy File, If Pattern.....	184
Delete File.....	186
Delete File with Force.....	186
Set Current Directory Working Path.....	187
Read Contents of File.....	187



TrimWhiteSpace from File Contents.....	187
Read File into Array.....	188
Create Folder.....	189
Detect if Folder is Empty.....	189
Detect if Folder Exists.....	190
Detect, Create or Delete Folder.....	190
Return File Version using Get-Package.....	191
Return File Version using FileVersionInfo/VersionInfo.....	191
Verify MD5 on File.....	192
Return MD5 Hash.....	192
Compare MD5 Hash.....	192
Unblock Downloaded Files.....	193
<b>PowerShell: Text File Operations.....</b>	<b>194</b>
Create Text File.....	194
Write to Text File.....	194
Append to Text File.....	194
Delete Text File.....	194
Read from Text File.....	194
Read from Text File, Line by Line.....	195
Rename Text Files, Replace Underscore.....	197
Sort Text File Ascending/Descending.....	199
Merge Two Text Files.....	201
Merge Two Text Files, Remove Dups.....	204
Return Specific Line in Text File.....	207
<b>PowerShell: Process Operations.....</b>	<b>210</b>
Start Process.....	210
Invoke-Expression.....	210
ScriptBlock.....	210
Start-Job.....	211
Stop Process: Graceful.....	211
Stop Process: Force.....	211
Detect if Process is Running.....	212
Change Process Base Priority.....	212
<b>PowerShell: Registry Operations.....</b>	<b>214</b>
Add Reg Key.....	214
Delete Reg Key.....	215



Delete Reg Key Value.....	215
Read Reg Key Value.....	217
Detect Reg Key, Return True or False.....	219
Detect Reg Key Data, Return True or False.....	220
Add Reg Access Rule.....	221
Add Reg Access Rule, Everyone, Full Control.....	222
Remove Reg Access Rule.....	223
Rebuild Reg SubKey ACLs.....	226
Return Reg AccessControlType.....	228
Change Reg Key Owner.....	229
Disable Reg Key Inheritance.....	229
Delete Reg Key User Principal.....	230
Return Reg SubKey Principals.....	231
Return Reg SubKey Principals Information.....	231
Add Reg Principal and ACE Rule.....	232
Update or Remove Reg ACE Rule.....	235
Add Reg Key for All Users.....	236
<b>PowerShell: Logical Constructs.....</b>	<b>239</b>
IF Condition.....	239
IF-ELSE Condition.....	240
IF-ELSEIF Condition.....	241
SWITCH Statement, Decision-making.....	242
FOR Statement, Loop.....	243
FOREACH Statement, Loop.....	244
WHILE Statement, Loop.....	245
TRY-CATCH Block, Exception Handling.....	246
<b>PowerShell: Data Structures.....</b>	<b>247</b>
Arrays.....	247
Stacks.....	249
Queues.....	251
Hash Tables.....	253
<b>PowerShell: Miscellaneous.....</b>	<b>255</b>
Create a Counter, For.....	255
Detect if 32 Bit or 64 Bit Environment.....	255
Create Event Log.....	255
Random Beeps.....	256



Add Sound.....	256
Using -NotLike to Filter Results.....	257
Return Chrome Version.....	257
Return SCCM Agent Version.....	258
Detect SCCM Agent.....	259
Detect FSLogix Environment.....	259
Detect AzureAd Device.....	259
Return Windows Build Version.....	260
Return BitLocker Passwords.....	260
Check Status of Service.....	261
Test Certificate Path & Set Certificate Location.....	261
Test if Certificate Exists.....	263
Delete Certificate Based on Thumbprint.....	263
Turn Off Monitors, Sleep Mode.....	263
PowerShell: Interactive Menu.....	264
PowerShell: Random Quotes.....	266
PowerShell: Message Box.....	268
<b>PowerShell: Forms.....</b>	<b>269</b>
Form with OK Button.....	269
Form with OK and Cancel Buttons.....	270
Opacity, Element Opacity.....	271
Form with OK, Cancel Buttons & Input Field.....	272
Common Form Options.....	274
Greenbar Animation.....	276
Green Scanner Animation.....	277
Red Scanner Animation.....	279
Fade Red to Black Animation.....	281
Fade In & Out Animation.....	283
Countdown 10-1.....	285
Percentage Counter 1-100%.....	287
Empty Reboot Form.....	289
<b>PowerShell: Task Scheduler.....</b>	<b>294</b>
On Demand App Install.....	294
On Demand App Repair.....	295
On Demand App Download.....	296
Create Scheduled Task to Run Process or Script.....	297



<b>7. AutoHotKey.....</b>	<b>298</b>
<b>AutoHotKey: Design.....</b>	<b>300</b>
Basic Construction.....	300
AutoHotKey: Mouse Block.....	301
AutoHotKey: Volume Control with Mouse Wheel.....	301
AutoHotKey: Window Always on Top.....	301
AutoHotKey: Caps Lock to Toggle Mute.....	302
AutoHotKey: Quick Launch Applications.....	302
AutoHotKey: Play a Tune.....	302
<b>8. AutoIt.....</b>	<b>304</b>
<b>AutoIt: Design.....</b>	<b>306</b>
Basic Construction.....	306
Create Text File.....	307
Create Folder.....	307
Copy File.....	307
Copy Folder.....	307
Delete File.....	308
Delete Folder.....	308
Open Notepad, Type 'Test'.....	308
AutoIt: Move Window Down.....	310
AutoIt: Move Window Up.....	311
AutoIt: Return Window Title.....	312
AutoIt: WinActivate, Take Control of Window.....	313
<b>9. Bash macOS.....</b>	<b>314</b>
<b>Bash: Design.....</b>	<b>316</b>
Basic Construction.....	316
<b>Bash: Script Execution.....</b>	<b>317</b>
Line Endings.....	317
Make Script Executable.....	317
Remove Executable Permission.....	317
<b>Bash: User Operations.....</b>	<b>318</b>
Return Logged in User.....	318
Create New Account.....	318
Delete Existing Account.....	319



Create Account, Set as Admin, and Show User.....	320
Copy File.....	322
Delete File.....	322
Copy Folder.....	322
Delete Folder.....	323
Change File/Folder Permissions.....	323
Mount & Attach DMG.....	323
Detach & Unmount DMG.....	324
Clear Intune Company Portal Cache.....	324
<b>Bash: Process Operations.....</b>	<b>325</b>
Open .App.....	325
Kill Process.....	325
<b>Bash: OS Operations.....</b>	<b>326</b>
Set Time & Date.....	326
Set Time Zone.....	326
Unload Service.....	326
<b>Bash: Miscellaneous.....</b>	<b>327</b>
Create Log File.....	327
Fix Sudoers Permissions Error.....	327
Sign PKG with Certificate.....	327
Check Signature on App.....	328
Install PKG Using Installer.....	329
Install Citrix Client from DMG.....	329
<b>Bash: Logic Examples.....</b>	<b>330</b>
IF Condition, Unblock Website.....	330
IF NOT Condition, Block Website.....	330
Logic Script.....	331
<b>10. Scripted Management: A.K.A. Automation.....</b>	<b>333</b>
Batch Script Examples.....	334
PowerShell Script Examples.....	336



## PROGRAMMING FUNDAMENTALS

<b>11. Foundational.....</b>	<b>342</b>
Classes.....	343
Properties.....	343
Methods.....	344
Inheritance.....	344
Encapsulation.....	345
Polymorphism.....	345
Abstraction.....	346
Constructors and Destructors.....	346
Static Members.....	347
Access Modifiers.....	347
Example Code.....	348
<b>12. C# Basics.....</b>	<b>351</b>
C#: Design.....	354
Basic Construction.....	354
C#: Return User UPN.....	356
C#: Return Username.....	359
C#: Return User SID.....	362
C#: Launch & Kill Process.....	364
C#: Read from Text File.....	365
C#: Read from Text File, Line by Line.....	365
C#: Write to Text File.....	367
<b>13. Logic.....</b>	<b>368</b>
Logical Operators.....	369
Common Logic Scenarios.....	370
Example Code.....	374
<b>14. Sorting Algorithms.....</b>	<b>376</b>
Quick Sort.....	377
Example Code.....	379
Merge Sort.....	383



Example Code.....	385
Heap Sort.....	390
Example Code.....	392
Insertion Sort.....	399
Example Code.....	401
Bubble Sort.....	406
Example Code.....	408
Counting Sort.....	412
Example Code.....	414
Selection Sort.....	418
Example Code.....	420
Radix Sort.....	424
Example Code.....	426
Bucket Sort.....	430
Example Code.....	432
Permutation Sort.....	437
Example Code.....	439

## UNIFIED SOLUTIONS

<b>15. Integration.....</b>	<b>444</b>
<b>16. Flowcharts for Code.....</b>	<b>446</b>
Flowchart Example 1.....	447
Flowchart Example 2.....	449
Flowchart Example 3.....	451
Flowchart Shapes.....	453
<b>17. Compilation &amp; Packaging.....</b>	<b>454</b>
Compiling Batch Scripts.....	456
Compiling PowerShell Scripts.....	458
Visual Guide to Compiling a Batch Script.....	464
Visual Guide to Compiling a PowerShell Script.....	468



<b>18. Deployment.....</b>	<b>480</b>
Deploying a Solution through SCCM.....	481
Deploying a Solution through Intune.....	485

## EXPANDED NOTES

<b>19. Power of the Command Line.....</b>	<b>491</b>
<b>20. Windows Console.....</b>	<b>493</b>
<b>Cmd: User Management.....</b>	<b>493</b>
Show User Accounts on Computer.....	493
Set User Account to Active.....	493
Set User Account Password.....	493
Set User Account to Never Expire.....	493
Show Users in Administrators Group.....	493
Add User to Administrators Group.....	494
Allow User to Change Time.....	494
Grant User Access to Files & Folders.....	494
<b>Cmd: Registry Management.....</b>	<b>496</b>
Take Everyone Ownership of Registry Key.....	496
Set Everyone Permissions on Registry Key.....	496
<b>Cmd: File &amp; Folder Management.....</b>	<b>497</b>
Copy Files & Folders.....	497
Delete Folders - Recursive.....	497
Take Ownership of Files & Folders.....	498
<b>Cmd: Operating System Management.....</b>	<b>499</b>
Reset Network Card.....	499
Reset Network Card, Full.....	499
Append TCP/IP DNS Suffix List.....	499
Enable/Disable Firewall.....	499
Disable Security Notifications.....	500
Reset Power Management.....	500
Delete Windows Service.....	500



Create a Windows Service.....	500
Import Certificate.....	500
Sign Executable with Certificate.....	501
Set Time Zone.....	501
Sync Time.....	501
Import Provisioned Package, AppxBundle.....	502
Import Provisioned Package, Cab.....	502
Import Provisioned Package, Gpedit.....	502
<b>Cmd: BitLocker Management.....</b>	<b>503</b>
Check BitLocker Status.....	503
Show BitLocker Password.....	503
Initialize TPM.....	503
Enable BitLocker.....	503
Suspend BitLocker.....	503
Resume BitLocker.....	503
<b>Cmd: Miscellaneous Commands.....</b>	<b>504</b>
Download Files from the Internet.....	504
Schedule Task.....	505
<b>21. PowerShell Console.....</b>	<b>506</b>
<b>Psh: Applications &amp; Packages.....</b>	<b>506</b>
Install MSIX Package.....	506
Remove MSIX Package.....	506
Install Provisioning Package.....	506
Remove Provisioned Package.....	506
Uninstall All Provisioning Packages.....	507
Repair Windows Apps.....	507
Install Package from Website.....	507
Install WinGet from PSGallery or Store.....	507
<b>Psh: Download Files from the Internet.....</b>	<b>508</b>
<b>22. Regular Expressions (RegEx).....</b>	<b>509</b>
<b>Example Code.....</b>	<b>510</b>
RegEx: Basic Match, Simple String Search.....	510
RegEx: Using '^' for String Start Matching.....	511
RegEx: Using '\$' for String End Matching.....	512
RegEx: Using '\d' to Match Digits.....	512



RegEx: Using '\w' to Match Word Characters.....	513
RegEx: Using '\s+' to Match Whitespace.....	514
RegEx: Using '\d+' to Return Phone Number.....	515
RegEx: Using '\w' & '\w+' to Return Email Address.....	516
RegEx: Using '\b\d{2}\d{2}\d{4}\b' to Match Date.....	517
RegEx: Using 'https?://[^s]+' to Match a URL.....	518
<b>23. What is the System Account?.....</b>	<b>519</b>
Leverage System Simulation.....	520
Testing in the System Account.....	521
<b>24. Formatted Timestamps.....</b>	<b>523</b>
Format Examples.....	524
Example Code.....	525
<b>25. Windows Commands &amp; Executables.....</b>	<b>526</b>
<b>26. Sysinternals Tools by Mark Russinovich.....</b>	<b>530</b>
<b>27. Atomic Series GitHub.....</b>	<b>534</b>



## 1. Greetings, Fellow Explorer!

---

The *Applied Atomics* book is a comprehensive learning guide for several scripting languages and the C# programming language. It is intended to provide practical examples, solutions for diverse tasks, and handy code snippets to tackle common challenges you're likely to encounter. Need to merge two files? It's here. Need to learn how to script? It's here. The content is organized by scripting languages, including Batch Script, PowerShell, AutoHotKey, AutoIt, and Bash, ensuring clarity and ease of navigation. Each section provides code for managing system operations, automating repetitive tasks, handling file manipulations, interacting with the registry, managing user desktops, and addressing other use cases you will most likely be confronted with in everyday scripting and administration. These are meant for your own testing and learning. The point of this book is to give you things to try, things to do that will enhance your own learning. Everyone always asks, "Where do I start?" You start here.

Whether you're a tech, system admin, automation engineer, or developer, the content is designed to empower you with the tools and knowledge needed to efficiently automate tasks and streamline workflows across diverse environments. By offering clear, step-by-step examples for both common and even some advanced tasks, it enables you to quickly implement effective solutions without needing to memorize intricate syntax or complex procedures. This is a book about doing—a book about achievement.

The foundational programming concepts explored in the latter sections provide deeper insights into programming, equipping readers with essential skills and a better understanding of logic and sorting algorithms. This guide is a valuable resource for the not-so-beginner users seeking a solid introduction and looking to expand their current expertise when creating automated solutions using code.

You will notice there are not overly verbose summaries for each block of code. I have chosen to fill these pages with code rather than lengthy explanations that often go unread. The included code comments are designed to be helpful and concise, of which, I have included many. I hope this format proves to be practical for learning and be a more effective use of your time. While I did not strictly write this for beginners, I do believe beginners will benefit from seeing code in action.



## Can you learn to do this? The answer is, Yes!

I know, I know, scripting and programming often seem like sorcery. So shrouded in mystery. So magical that only the dev wizards can truly grasp the complexity. However, at their core, scripting and programming are about understanding and applying fundamental concepts to solve problems and automate tasks.

Think of it like building a house: You start with a strong foundation, and then add layers of complexity and functionality. Just like any other skill, scripting and programming start with the basics. These include understanding variables, data types, functions, and control structures like loops and conditionals. These elements, or atomic structures (atomics), are the building blocks of any script or program. Combining these basic components and building upon them can create very powerful and practical solutions. That's really what this book is about—seeing the building blocks repeated over and over again.

But what exactly are variables and data types, or...these control structures and functions I mentioned?

- Variables are like storage bins that hold data. This bin may have a name. That bin may have a place.
- Data types define the kind of data a variable can hold, such as integers, strings, or Booleans. In our bin analogy, specific bins hold specific data types.
- Control structures are the instructions that manage the flow of a program. Loops (`for`, `while`) allow you to repeat actions, while conditionals (`if-else`) let you make decisions based on certain conditions. You're going to see a lot of those in this book.
- Functions are just reusable blocks of code that perform a specific task. They help in organizing code and making it more modular and maintainable.

Okay. That is fine and all, but how does one actually script or code? How exactly do you learn to accomplish things with code?

It is all about *need* and understanding where *automation* can help you.

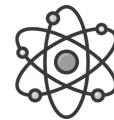


For me, automation is where the real magic happens. This technical skill, rooted in software development—and the automation mindset—extends far beyond the roles of a technician, system administrator, or even a developer. Oftentimes, our jobs create silos of knowledge bases with their respective skill sets. Career training focuses on specific product lines with textbook-style support. The problem is, our world is not a textbook. Most companies operate in a state of controlled chaos, not in some sterile product lab. If you are like me, you have witnessed vendor apps that have failed, need repair, need upgrading, need to be rolled back, or require some other *undocumented*, custom solution. If you contact the vendor, you receive crickets, or some form of delayed support, or my favorite: "That cannot be done." It can be done; I assure you it can.

This is where the *need* part comes in. We need solutions that can address vendor gaps in support and features, while making our lives a little bit easier every day. Code is here to help you. Automation is here to help you.

## Where to Begin?

- **Step 1** | You want a lab or computer you can experiment on. Running untested code in a production environment can be dangerous—to networks, to data, and to your employment status. Build a lab.
- **Step 2** | You want to install the tools you will need to begin coding and testing code. I recommend Notepad++ for the scripting languages, and Visual Studio Community for C#. You will find other tools and resources listed in this training material as well.
- **Step 3** | Set aside time for learning. You need time to test and experiment with code. Type the code. It is part of the learning process.
- **Step 4** | Enhance your existing skills with this training material. Begin testing each segment of code, making your own changes, and building upon what you already know. Think of creative ways to improve upon the code examples. Ask yourself, "How does this code work? What does it do?"
- **Step 5** | Save your scripts, especially ones you have tested and know they work. You will be surprised how often you will come back to them. I have scripts from decades ago in my personal archive. Some code snippets are in this book.



By building upon the concepts of atomic structures (really dig into how they work), and chaining together scripts, programs, and services, you can automate complex workflows, save time, and reduce errors. Scripting languages like Batch, PowerShell, and Bash (and AutoIt & AutoHotkey) are designed for automating tasks and managing systems. Scripting languages are often easier to learn and use than full-fledged programming languages, like C#. The most important thing to remember is that bricks build castles, meaning the smallest building blocks of code can eventually lead to some impressive solutions—a little bit of scripting knowledge goes a long way. When you are working in the lab, think about how segments of code can be *chained* together or *integrated* with other code to accomplish the things you want. This technique alone can enable you to do much more, much faster.

The journey from mystery to mastery is about continuous learning and practice. Embrace the basics, experiment with different languages (use this book as a start), begin to see the common structural components and core mechanics that apply to all code, and you will soon be creating your very own magical solutions. And I'm not joking about the magical part. To most people, automation is nothing short of magic.

**"Any sufficiently advanced technology is indistinguishable from magic."**

**Arthur C. Clarke**



## So critical, it requires its own page.

**The most important thing you will ever learn: Security Context.**

---

Understanding the security context in which code is executed is a requirement; it is not optional. Always evaluate whether the context is that of a **restricted user**, an **administrator**, or the **system account**. Deliberately test and validate your code's behavior across each context to ensure it produces the intended results. Scripts and apps that must run in the system account, but access the primary user's processes, files and folders, and registry hive, have to be specifically written for that accessibility. Simply capturing an environmental variable or referencing HKEY\_CURRENT\_USER in the registry isn't going to work. Not at all. Surprisingly, even some of the largest organizations, with trillion-dollar valuations, have yet to fully grasp and implement this essential practice.

See 'What is the System Account?' in Expanded Notes for more.

---



## 2. Introduction

---

When I started writing this book, it was just a scrappy little document—a cheat sheet for myself, really. Thirty-plus pages of code snippets thrown together over the years, surviving only because they were occasionally too useful to delete. It was around Christmas 2024, when I found myself in a festive lull, that I decided to clean up the doc. Formatting code snippets during the holidays—yeah, I know, so thrilling. But somewhere between fixing indentations and debating the merits of camelCase (you either love it or hate it), it struck me: Why not turn this into something more? After all, I've spent decades in the trenches of scripting and programming. Why not share at least some of that experience?

The problem is, I loathe the typical scripting book format. Have you ever noticed how many of the books are just glorified help files? Sure, the books walk you through some basics (I guess), but the writing often feels detached. And don't get me started on the endless walls of black-and-white text, where syntax highlighting is apparently a lost artform (No, seriously. Where is the syntax highlighting?). The result is a dry, joyless slog through 1,200 pages of the Sahara Desert. So dusty. So dry. To future authors, syntax highlighting provides useful hints to how code works. Stop skipping it.

This book is my *first* attempt to do better. I am no author (let's be clear about that), but have been known to create some pretty decent technical docs. With a book, I wanted something intuitive and high-quality, but also lightweight—less about drowning you in theory and reference material, and more about the fusion of code with learning material you can actually use, and actually want to use. A book where code takes center stage. It's built around the idea that understanding *how* and *why* things work is just as important as *what* works and *seeing* it work. I've always loved the atomic elegance of data structures and algorithms (the very building blocks of all code), but I've also spent many years using code to solve real-world problems. So, *how does the not-so-beginner continue to learn and build their skills?* That was the question and premise that drove the creation of this content.

Here's the result: A book that combines programming principles and techniques, with the pragmatism of scripted solutions. It's designed to show you how to write code that actually works, do so as quickly as possible, and to understand the power of code.



## What This Book Is

---

- **Applied Solutions:** Demonstrations of programming principles with real-world relevance.
- **A Training Guide:** Think of this as a cross between a college workshop and a code repository. That's my objective.
- **Code-First:** Each section prioritizes working examples. Any descriptions or comments are concise and focus on how the code works.
- **Language-Agnostic at its Core:** While examples may favor certain languages, the principles are *universal*, which is exactly the point. Remember this universal theme throughout all code examples.
- **Focused on Intermediate Users:** The content is for those who have at least some of the basics down and looking for the next step.

## What This Book Isn't

---

- **A Reference Manual:** If you need exhaustive syntax lists, command flags, or detailed documentation, you're better off with official resources. Those books serve their purpose; this just isn't one of them.
- **An Encyclopedia:** This book doesn't try to cover every programming language or scripting environment.
- **A History Book:** I love programming history, but this isn't the place for lengthy history lessons. Any history of a language will be brief, and then the code starts.
- **A Theoretical Deep Dive:** There's just enough theory to give context to the code, but hopefully you won't get lost in a maze. I want to give you enough theory to ask, "What comes next?"
- **Fluff:** No filler content, unnecessary verbosity, or tangential musings. I personally love this lighter format. Maybe you will too.



### 3. Definitions

---

#### Data Structures

---

What exactly are data structures? At their core, data structures are sophisticated ways of organizing and storing data in a computer, and enabling efficient access. They serve as the building blocks (or atomic structures) of computer science, laying the foundation for how we manage vast amounts of information systematically and effectively. With the right choice of data structure, developers can dramatically optimize programs in terms of speed, memory usage, and scalability, transforming complex problems into manageable solutions.

From arrays and linked lists to stacks, queues, hash tables, trees, and graphs, each data structure is designed to excel at specific tasks, whether it's searching, sorting, or managing relationships. Mastering these structures not only hones problem-solving skills, but also cultivates the ability to write cleaner, more efficient, and highly scalable code. By understanding their nuances, developers unlock the potential to create robust software capable of handling the most demanding computational challenges.

Data structures don't exist in isolation; they are the framework upon which algorithms operate. Algorithms use data structures as tools to solve problems, and the effectiveness of an algorithm often hinges on the underlying structure. For example, while a hash table might be ideal for lightning-fast lookups, a binary search tree could be better suited for maintaining sorted data. The synergy between algorithms and data structures underscores their importance in fields ranging from artificial intelligence to database systems.

Learning data structures isn't just about improving technical skills—it's about developing the mindset to solve real problems with precision and creativity.

It's a universal thought process for efficiency.



## Common Data Structures

- **Array:** A collection of elements stored in contiguous memory locations.
- **Linked List:** A sequence of nodes where each node points to the next, enabling dynamic memory allocation.
- **Stack:** A collection of elements that follows the Last In, First Out (LIFO) principle.
- **Queue:** A collection of elements that follows the First In, First Out (FIFO) principle.
- **Hash Tables:** Uses a hash function to compute an index into an array of buckets or slots, from which the desired value can be found.
- **Heaps:** Max-Heap—The value of each node is greater than or equal to the values of its children (the largest value is at the root). Min-Heap—The value of each node is less than or equal to the values of its children (the smallest value is at the root).
- **Tree:** A hierarchical data structure with nodes connected by edges, starting from a root node.
- **Graph:** A set of nodes (vertices) connected by edges, which can be directed or undirected.



## Algorithms

---

Algorithms are step-by-step instructions for solving problems or performing tasks, forming the backbone of computer science. Sorting and searching algorithms, such as quicksort, mergesort, and binary search, are essential for organizing and retrieving data efficiently. Recursion, a powerful programming technique, allows functions to call themselves to solve complex problems by breaking them into simpler sub-problems. Together, these concepts underpin countless technologies, like web, cybersecurity, streaming, and encryption, just to name a few. Algorithms can be very powerful.

### Basic Algorithms

- **Sorting:** Arranging elements in a specific order (ascending or descending).
- **Searching:** Locating a specific element within a dataset.
- **Recursion:** Solving a problem by breaking it down into smaller instances of the same problem.

### Key Terms

- **Time Complexity:** Measures how the runtime of an algorithm changes as the input size increases.
- **Space Complexity:** Measures the memory an algorithm needs relative to the input size.
- **Big O Notation:** Describes the worst-case performance of an algorithm, showing how it scales with input size.
- **Recursion:** A programming technique where a function calls itself to solve smaller instances of a problem.
- **Divide and Conquer:** Breaks a problem into smaller sub-problems, solves them individually, and combines the results.
- **Brute Force:** Tries all possible solutions to a problem, ensuring the correct answer but often inefficient.



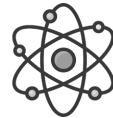
## Search Algorithms

---

Search algorithms are techniques used to locate specific data within a collection, such as an array, list, or database. They range from simple methods like linear search, which checks each element sequentially, to advanced approaches like binary search, which efficiently narrows down possibilities in sorted data. More complex algorithms, such as depth-first and breadth-first search are designed for traversing graphs and trees. Mastery of search algorithms is crucial for optimizing performance in everything from database queries to AI pathfinding.

### Common Search Algorithms

- **Quick Sort:** Divides the dataset using a pivot, then recursively sorts the partitions.
- **Merge Sort:** Recursively divides the dataset into halves, sorts them, and merges them back together.
- **Heap Sort:** Uses a binary heap to repeatedly extract the maximum or minimum element and rearrange the dataset.
- **Insertion Sort:** Builds the sorted dataset one element at a time by placing elements in their correct position.
- **Bubble Sort:** Repeatedly swaps adjacent elements if they are in the wrong order.
- **Counting Sort:** Counts the frequency of each element and uses it to sort the dataset.
- **Selection Sort:** Repeatedly selects the smallest (or largest) element and places it in the sorted portion.
- **Radix Sort:** Sorts numbers digit by digit, starting from the least significant digit.
- **Bucket Sort:** Divides the dataset into buckets and sorts each bucket individually.
- **Permutation Sort:** Generates all permutations of the dataset and selects the sorted one.



## Data Structures vs. Algorithms

---

### So, What's the Difference Between Data Structures and Algorithms?

Data structures focus on how data is stored, organized, and accessed, while algorithms define the steps to solve problems or manipulate that data. They complement each other: data structures provide the foundation, and algorithms provide the means to operate on them. An efficient algorithm relies on the right data structure, and vice versa. When paired, they form the backbone of effective programming and problem-solving.

Aspect	Data Structures	Algorithms
<b>Definition</b>	The way data is organized, stored, and retrieved.	A set of instructions used to solve a specific problem or perform a specific task.
<b>Purpose</b>	Efficiently organizes and stores data for easy access.	Provides a systematic approach to solving problems.
<b>Operations</b>	Insertion, Deletion, Search, Update, Traverse, etc.	Sorting, Searching, Optimization, Pathfinding, etc.
<b>Importance</b>	Essential for efficient data management and access.	Crucial for developing optimized software solutions.
<b>Relationship</b>	Provides a framework for algorithms to operate on.	Operates on data structures to process or manipulate data.
<b>Performance</b>	Determines the efficiency of the algorithms that use them.	Impacts the performance of the software solution overall.
<b>Examples</b>	Array, Linked List, Stack, Queue, Tree, Graph, Hash Table.	Sorting, Searching, Dynamic Programming, Divide & Conquer.



## Loops in Context

---

Loops are control flow structures, bridging data structures and algorithms. They enable the practical execution of concepts, transforming abstract structures into actionable code and ensuring that algorithmic logic can be implemented in an efficient, repeatable manner. Are you seeing how everything is slowly becoming connected, and is essential to one another? You're witnessing the very infrastructure of computer science starting to take form. You'll notice a frequent use of loops in our code examples.

The concept and structure of loops are essential and crucial to understand. Mastering them will set you on the path to becoming a true expert in the craft.

Aspect	Data Structures	Algorithms
Purpose	Iterating through elements, managing traversal.	Repeating steps to solve problems or achieve results.
Examples	Traversing arrays, linked lists, stacks, queues, trees, and graphs.	Sorting (Ex: Bubble Sort, Quick Sort), searching (Ex: Binary Search), and optimization tasks (Ex: Dijkstra's).
Common Types	for, while, foreach for access and updates.	for, while, recursion for iterations and backtracking.
Complexity Focus	Focus on linear, tree-like, or graph traversal patterns like Breadth-First Search (BFS) and Depth-First Search (DFS).	Focus on reducing time/space complexity (Ex: dynamic programming, divide-and-conquer).
Efficiency Metrics	Access time ( $O(1)$ for arrays, $O(\log n)$ for trees), traversal efficiency ( $O(n)$ for linked lists).	Number of iterations, convergence rate, recursion depth, or branching factor.
Key Challenges	Handling circular references, avoiding infinite loops in cyclic structures.	Preventing stack overflows (in recursion), ensuring termination, and avoiding redundant computations.



## Scripting Language Concepts

---

Scripting concepts lay the groundwork for automating tasks and crafting dynamic workflows across various environments. They include essentials like variables for storing data, control structures for decision-making and iteration, and functions for creating reusable code blocks. Scripting languages shine in areas like text manipulation, file handling, and interacting with operating systems, making them perfect for simplifying repetitive tasks. Features such as error handling, regular expressions (we'll get into those), and libraries contribute to building robust and scalable solutions. With dynamic typing and interpreted execution, languages like PowerShell, Batch, Python, JavaScript, and Bash provide the versatility and ease needed to streamline complex operations with efficiency and precision.

Category	Description & Examples
<b>Variables</b>	Used to store data. Ex: set "name=John"; in Batch, \$name = "John"; in PowerShell.
<b>Data Types</b>	Types of data: integers, strings, arrays, objects. Ex: "text", 123, [1, 2, 3].
<b>Control Structures</b>	Loops (for, while), conditionals (if-else, switch). Ex: if (\$x -eq 5) { ... } in PowerShell.
<b>Functions</b>	Reusable blocks of code. Ex: function Get-Name { ... } in PowerShell, :Label in Batch.
<b>Error Handling</b>	Managing runtime issues. Ex: try { ... } catch { ... } in PowerShell, If conditional and return codes in Batch.
<b>Input/Output</b>	Reading user input and displaying output. Ex: read-host in PowerShell or set /P name= in Batch.
<b>Regular Expressions</b>	Used for pattern matching in strings. Ex: \$_ -match "\d{3}-\d{2}-\d{4}" in PowerShell.
<b>File Handling</b>	Reading/writing to files. Ex: Get-Content or Out-File in PowerShell.
<b>Scripting Environment</b>	Interaction with the OS. Ex: \$env:PATH in PowerShell, %computername% in Batch..



<b>Modules/Libraries</b>	Reusable code packages. Ex: Import-Module in PowerShell.
<b>Comments</b>	Notes for code. Ex: # Comment in PowerShell, REM or :: in Batch.
<b>Arrays/Collections</b>	Managing lists of data. Ex: \$array = @(1,2,3) in PowerShell.
<b>Automation Features</b>	Task-specific tools. Ex: Invoke-WebRequest in PowerShell or curl in shell scripting.
<b>Dynamic Typing</b>	Variables can hold different types at runtime. Ex: \$x = 5; \$x = "text"; in PowerShell.
<b>Interpreted Execution</b>	Code runs without compilation. Ex: PowerShell scripts are executed directly. PowerShell uses powershell.exe, Batch uses cmd.exe.

With scripting, our primary focus will be on Batch and PowerShell (a bit of bash as well), but the core concepts we cover are applicable to nearly all interpreted languages. By mastering these fundamentals, you can significantly enhance your scripting skill set and adaptability across all code.

---

While this book doesn't focus entirely on data structures, algorithms, or the deep theoretical exploration thereof, these ideas fill the pages of this book. After all, you can't script or program effectively without relying on the foundational principles of computer science. When you see loops, conditional logic, or arrays, understand, we are walking through the halls of data structures and algorithms.

My aim here is to show you practical solutions and techniques, code that works, and ideas you can see in action.

---

If nothing else, I hope the content within these pages ignites your curiosity and motivates you to continue exploring and learning.

---

**"If you love something you can put beauty into it." -Donald Knuth**



## 4. Important Links

---

The books, tools, and websites listed here are valuable resources to support your learning and skill development. While they don't cover everything available, they provide more than enough guidance to help you advance to the next level.

### Books

- An Illustrated Guide for Programmers and Other Curious People
- C# Illustrated C# 7, 5th Edition
- Introduction to Algorithms 3rd Edition
- Mastering PowerShell Scripting: Automate and Manage
- PowerShell In Depth
- Windows-Command-Line-Administration-Instant-Reference
- Windows Internals Developer Reference, Part 1 | Part 2
- Troubleshooting with the Windows SysInternals Tools
- The macOS User Administration Guide
- Learning the bash Shell: Unix Shell Programming
- PowerShell for Sysadmins: Workflow Automation Made Easy
- Learn PowerShell in a Month of Lunches
- PowerShell Cookbook: Your Complete Guide to Scripting



## Tools

- AutoHotKey Scripting Tools (Code Editor)
- AutoIt Scripting Tools (Code Editor)
- Notepad++ (Code Editor)
- Orca MSI Editor
- PrimalScript Scripting IDE (Code Editor)
- Sysinternals Advanced Utilities, by Mark Russinovich
- Visual Studio Community Edition IDE
- WinRAR Compression Software (Used to package resources and make EXEs)
- PS2EXE-GUI (TechNet-Gallery)
- PS2EXE Module (PowerShell Gallery)

## Websites

- Advanced Tools & Scripting with PowerShell, by Jeffrey Snover
- Apple Developer
- Microsoft Developer
- About the Windows Registry
- Windows Registry Information for Advanced Users



## SCRIPTING LANGUAGES



### 5. Batch Script

---

#### Brief Overview

**Batch** programming traces its roots back to the rugged days of MS-DOS in the 1980s. Back then—before a structured file format would appear—typing out the same commands over and over was as thrilling as watching paint dry. Ah, so many memories. Enter the batch file—a no-frills solution to automate command-line tasks, one instruction at a time. The *Batch* name? A nod to "batch processing," a way to execute a series of commands without babysitting the keyboard.

As MS-DOS morphed into Windows (which would take Microsoft many years to escape the perception that DOS was the primary OS), Batch scripting evolved but stayed true to its roots: simplicity and compatibility. Batch became the go-to tool for system admins and IT pros who wanted quick automation without the overhead of learning complex languages.

It's where I got started in the professional world. I worked for an ISP in the 90s (Panhandle Online), when ISPs were just starting to roll out Internet service to the world. I programmed Batch scripts to automate the dial-in process. It was at this time that the seed of automation was planted, I just didn't realize it at the time. Of course, modern contenders like PowerShell now dominate the scripting scene, but Batch files remain the fierce underdog—reliable, easy to use, and always ready to get the job done. I still love and use it.

Batch scripting, while often considered old school, has a timeless quality that makes it just as relevant today, especially when efficiency is key. It's not flashy, but it gets results—and sometimes, that's all you really need.



## Why is it still practical today?

- **Universal Compatibility:** Batch scripts work out-of-the-box on all Windows systems without additional installations.
- **Simplicity:** The syntax is easy to learn, making it a great starting point for beginners in scripting.
- **Legacy System Management:** Many organizations and users still rely on legacy systems where batch scripts excel.
- **Quick Automation:** It is ideal for simple tasks like file, process and user management, software deployment, or running scheduled jobs.
- **Integration:** Batch files can invoke external programs and scripts, allowing integration with modern tools.

Despite some of its limitations—like weaker error handling and less robust functionality compared to PowerShell—Windows Batch programming remains a practical and efficient tool for quick automation and managing Windows environments. Its enduring presence underscores its value as a lightweight, reliable solution in the toolkit of IT professionals. And, I'm not going to lie, there is just something nostalgic about using it.



## Caveat

**.BAT** and **.CMD** files are both script files used for executing commands in the Windows command prompt, but they have a subtle difference in behavior:

- **.BAT** files: These are the older script format, originating from MS-DOS. They run in a DOS-like environment, and any commands within them are executed sequentially. They are backward-compatible with older versions of Windows.
- **.CMD** files: These are used in more modern Windows environments (starting with Windows NT). The key difference is that .CMD files are designed to handle certain command-line behaviors differently, like how error handling is managed. For example, in a .CMD file, the CALL command inside a block can affect the command prompt's error level, which behaves differently than in a .BAT file.

In essence, both serve the same purpose, but .CMD files are preferred in newer Windows versions due to their expanded handling of commands. So, when you save your batch files, save them with the .CMD extension.

## Examples

- Script.cmd
- My\_Script.cmd
- Copy-Script.cmd



## Batch: Design

---

[https://en.wikibooks.org/wiki/Windows\\_Batch\\_Scripting](https://en.wikibooks.org/wiki/Windows_Batch_Scripting)

### Basic Construction

---

Here we are; here we go. Let's build our first **Batch** script. Using your favorite code editor (Notepad++ is a good one to start with), *type* and then *save* this code as *script.cmd* and *run* it (double-click it). Read the code comments for details, which is usually where I live.

```
:: Disables the display of commands as they are executed.  
@ECHO OFF  
  
:: Set the window title.  
TITLE Greeting Script  
  
:: Set text and background color.  
COLOR 0A  
  
:: Clears the console screen.  
CLS  
  
:: Prompt the user for their name. How to accept input.  
SET /P "Name=What is your name? "  
  
:: Display a personalized greeting.  
ECHO Hello, %Name%! Have a great day!  
  
:: Add timing to our script. You will often need to wait for things to happen.  
TIMEOUT /T 4 >nul  
  
:: Pause the script to allow the user to see the message.  
PAUSE  
  
:: Create a log file.  
ECHO Success! Basic Construction Test > log.txt  
  
:: Exit the script with success code.  
EXIT /B 0
```



## Batch: Enable Delayed Expansion

### ENABLEDELAYEDEXPANSION

**SetLocal EnableDelayedExpansion** allows you to use variables inside loops or conditional blocks where their values can change dynamically during the execution of the script. The caveat of this command is that percent signs become *exclamation points*.

```
@ECHO OFF

SET Counter=0
SET Result=

SETLOCAL EnableDelayedExpansion
FOR %%I IN (1 2 3) DO (
    SET Counter=%I
    SET "Result=!Result!!Counter! "
)
ECHO !Result!
ENDLOCAL

SET Counter=0
SET Result=

:: This is where we'll see if the delayed expansion is working.
FOR %%I IN (1 2 3) DO (
    SET Counter=%I
    SET "Result=%Result%Counter% "
)
ECHO %Result%
PAUSE
```

### | output |

With EnableDelayedExpansion statement: 1 2 3 | This is what we wanted.

Without EnableDelayedExpansion statement: 0 | Our loop didn't produce the results we wanted.



## Batch: String Operations

---

### Remove Leading Space

---

#### ENABLEDELAYEDEXPANSION

Uses delayed variable expansion to remove the leading space from the String variable.

" TestText" becomes "TestText"

```
@ECHO OFF

SETLOCAL EnableDelayedExpansion
SET "String= TestText"
ECHO Before: "%String%"

:: Look, we now have exclamation points. Do you remember why?
:: Modify String
SET "String=!String:~1!"
ECHO After: "%String%"
ENDLOCAL

PAUSE
```

### Remove Trailing Space

---

#### ENABLEDELAYEDEXPANSION

Uses delayed variable expansion to remove the trailing space from the String variable.

"TestText " becomes "TestText"

```
@ECHO OFF

SETLOCAL EnableDelayedExpansion
SET "String=TestText "
ECHO Before: "%String%"

:: Modify String
SET "String=!String:~0,-1!"
ECHO After: "%String%"
ENDLOCAL

PAUSE
```



## Detect & Remove Leading Space

-----  
**BOOLEAN | ENABLEDELAYEDEXPANSION | FOR**

Our previous space removal code can be elevated with *Boolean* logic. A Boolean expression is a particular case of an integer expression that returns either **TRUE** or **FALSE**. Employing logic in any form enhances flexibility in processes, control, and output. While Batch programming doesn't have true Boolean functionality, the expressions can be simulated and used quite well.

```
@ECHO OFF
SETLOCAL EnableDelayedExpansion

:: Define Variables
SET "String= TestText"
ECHO Before: "%String%"
:: Default Bool Value
SET "SPACE_DETECTED=False"

:: Loop Through Characters
:: Using a for loop to iterate through the string characters.
FOR /L %%I IN (0,1,255) DO (
    SET "CHAR=!String:~%%I,1!"
    IF "!CHAR!"=="" GOTO :CHECK_DONE
    IF "!CHAR!"==" " (
        SET "SPACE_DETECTED=True"
        GOTO :CHECK_DONE
    )
)

:CHECK_DONE
:: TRUE or FALSE result
ECHO Space Detected: %SPACE_DETECTED%

:: Modify String (remove spaces)
SET "String=!String: =!"
ECHO After: "%String%"

:: Why does this pause look different? We are chaining commands.
:: Here, we connect a newline with pause. You try it.
:: Every language has the ability to chain commands and statements together.
ENDLOCAL & ECHO. & PAUSE
EXIT /B 0
```



## Replace Underscore with Hyphen

### ENABLEDELAYEDEXPANSION

Replace an underscore with a hyphen in a string.

```
@ECHO OFF  
  
SETLOCAL EnableDelayedExpansion  
  
SET "String=Test_Text"  
ECHO Before: "%String%"  
  
:: Replace the underscores with hyphens.  
SET "String=%String:_=-%"  
  
ECHO After: "%String%"  
ENDLOCAL  
  
ECHO. & PAUSE
```

## Create Formatted Timestamp

### DATE | FOR | TIME

We use FOR loops and two system files, DATE.exe & TIME.exe, to create a custom timestamp, YYYYMMDD\_HHMM. You can use FOR loops to capture the output of commands, and then parse that content into something useful inside your script.

```
@ECHO OFF  
  
:: Get Current Date  
FOR /F "tokens=2 delims= %%A IN ('DATE /T') DO SET "currDate=%%A"  
  
:: Rearrange the date components (year, month, day) to create a timestamp.  
FOR /F "tokens=1-3 delims=/-. %%A IN ("%currDate%") DO SET  
"timestamp_date=%%C%%A%%B"  
  
:: Get Current Time  
FOR /F "tokens=1-2 delims=:: %%A IN ('TIME /T') DO SET "currTime=%%A%%B"  
  
:: Parse the time format to remove any trailing AM/PM.  
FOR /F "tokens=1-2 delims= %%A IN ("%currTime%") DO SET "timestamp_time=%%A"
```



```
:: Combine the formatted date and time.  
SET "timestamp=%timestamp_date%_%timestamp_time%"  
  
:: Show Timestamp  
ECHO Our formatted timestamp: %timestamp%  
  
ECHO. & PAUSE  
  
ECHO /B 0
```

**See:** Section 24. Formatted Timestamps



## Batch: User Operations

---

[CALL](#) | [FIND](#) | [FINDSTR](#) | [FOR](#) | [REG](#)

How to test from the System Account ([PsExec](#)):

```
PsExec64 -S CMD  
"C:\Batch\script.cmd"
```

### Return User SID

#### Method 1

---

**Scope:** Admin, System Account, PC, VDI

Retrieves the user SID from the LoggedOnUserSID reg key.

```
@ECHO OFF  
  
:: Define Variables  
SET  
"regPath=HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Authentication\LogonUI\SessionData"  
  
SET "regKey=LoggedOnUserSID"  
  
:: Query the root hive, and then look for the specified reg key.  
FOR /F "tokens=* %%A IN ('REG QUERY "%regPath%" /S /REG:64 2>nul ^| FINDSTR  
/R /C:"HKEY_LOCAL_MACHINE") DO (  
    FOR /F "tokens=3 %%B IN ('REG QUERY "%A" /V "%regKey%" /REG:64 2>nul  
    ^| FIND "REG_SZ") DO (  
        SET "uSID=%%B"  
    )  
)  
  
ECHO %uSID%  
  
ECHO. & PAUSE
```



## Method 2

---

**Scope:** Admin, System Account, PC, VDI

Retrieves the user SID from the SelectedUserID reg key.

```
@ECHO OFF

:: Query the reg key and set and display the user SID.
FOR /F "tokens=3" %%B IN ('REG QUERY
"HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Authentication\LogonUI" /V
"SelectedUserID" /REG:64 2^>nul ^| FIND "REG_SZ"') DO (
    SET "uSID=%%B"
)

ECHO %uSID%

ECHO. & PAUSE
```

## Method 3

---

**Scope:** Admin, System Account, PC

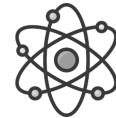
Retrieves the user SID from the LastLoggedOnUserID reg key.

```
@ECHO OFF

:: Query LogonUI subkey, LastLoggedOnUserID reg key and return the user SID.
FOR /F "tokens=3" %%C IN ('REG QUERY
"HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Authentication\LogonUI" /V
"LastLoggedOnUserID" /REG:64 2^>nul ^| FIND "REG_SZ"') DO (
    SET "uSID=%%C"
)

ECHO %uSID%

ECHO. & PAUSE
```



## Method 4

**Scope:** Admin, System Account, PC

Retrieves the most recent logged-on user session from the registry and extracts the user's Security Identifier (SID). It navigates through session data, identifies the latest session key, and fetches the LoggedOnUserID value. If successful, it displays the SID; otherwise, it outputs an error message indicating the absence of a valid session or key.

```
@ECHO OFF
SETLOCAL EnableDelayedExpansion

SET
"regPath=HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Authentication\LogonUI\SessionData"
SET "regKey=LoggedOnUserID"

:: Retrieve Latest Session
FOR /F "tokens=* %%I IN ('REG QUERY "%regPath%" /REG:64 2^>nul ^| FINDSTR /R
/C:"\\[0-9]*$") DO (
    SET "lastSession=%~NXI"
)

:: Check Session
IF NOT DEFINED lastSession (
    ECHO Error: No session found.
    EXIT /B 1
)

:: Retrieve the User SID from the latest session.
FOR /F "tokens=2,* delims= " %%I IN ('REG QUERY "%regPath%\!lastSession!" /V
%regKey% /REG:64 2^>nul') DO (
    SET "value=%~J"
)

:: Parse content if UPN is detected. Drops the domain name.
IF DEFINED value (
    ECHO User SID: !value!
) ELSE (
    ECHO Error: LoggedOnUserID key not found.
)
ENDLOCAL
ECHO. & PAUSE
```



## Method 5

**Scope:** Admin, System Account, PC, VDI

Review this. Try to understand how powerful it is. The CALL to GETREG gives us the ability to scan through hive keys until we find a specific reg key. There is no need to know exactly how many subkeys there may be. It searches until it finds the value, assuming there is one.

```
@ECHO OFF
:: Define Variables
SET
"regPath1=HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Authentication\LogonUI\SessionData"

SET "regValue1=LoggedOnUserSID"
SET "regHive1=HKEY_LOCAL_MACHINE"
SET "retValue1=uSID"
SET "regType1=REG_SZ"

:: See how we are passing config items to our function?
:: This creates a dynamic block of code we can use for other reg keys.
CALL :GETREG "%regPath1%" "%regValue1%" "%regHive1%" "%retValue1%"
"%regType1%"

ECHO %uSID%

ECHO. & PAUSE & EXIT

:: Our Function
:: This is where the magic happens. Registry keys are being scanned.
:GETREG
SET "regPath=%~1"
SET "regValue=%~2"
SET "regHive=%~3"
SET "retValue=%~4"
SET "regType=%~5"

FOR /F "tokens=*" %%A IN ('REG QUERY "%regPath%" /S /REG:64 2>nul ^| FINDSTR
/R /C:"%regHive%"') DO (
    FOR /F "tokens=3" %%B IN ('REG QUERY "%A" /V "%regValue%" /REG:64 2>nul
    ^| FIND "%regType%"') DO (
        SET "%retValue%=%B"
    )
)
GOTO :EOF
```



## Return Current User

### FOR | TASKLIST

**Scope:** Admin, System Account, PC, VDI

Retrieves the username of the user running explorer.exe by parsing the output of the TASKLIST command, splits the result into the domain and username, and displays the username.

```
@ECHO OFF

FOR /F "tokens=1,2,*" %%A IN ('TASKLIST /FI "IMAGENAME eq explorer.exe" /FO
LIST /V') DO (
    IF /I %%A %%B=="User Name:" SET "domainUser=%%C"

    FOR /F "tokens=1,2 delims=\\" %%A IN ("%domainUser%") DO SET "Domain=%%A" & SET
    "User=%%B"

    ECHO %User% & ECHO. & PAUSE
```

## Return Current User Domain

### FOR | TASKLIST

**Scope:** Admin, System Account, PC, VDI

Retrieves the username from the explorer.exe process by parsing the output of the TASKLIST command, splits the result into the domain and username, and displays the domain.

```
@ECHO OFF

FOR /F "tokens=1,2,*" %%A IN ('TASKLIST /FI "IMAGENAME eq explorer.exe" /FO
LIST /V') DO (
    IF /I %%A %%B=="User Name:" SET "domainUser=%%C"

    FOR /F "tokens=1,2 delims=\\" %%A IN ("%domainUser%") DO SET "Domain=%%A" & SET
    "User=%%B"

    ECHO %Domain% & ECHO. & PAUSE
```



## Return UPN

[FIND](#) | [FINDSTR](#) | [FOR](#) | [REG](#) | [UPN](#)

### Method 1

**Scope:** Admin, System Account, PC, VDI

Returns the UPN. A User Principal Name (UPN) is the name of a user in an email address-like format used to uniquely identify a user in a network or directory service, such as Microsoft Active Directory (AD). It typically takes the form: username@domain.com. You probably didn't realize you already knew what a UPN was.

```
:: Compact Code
@ECHO OFF

SETLOCAL EnableDelayedExpansion

SET
"regPath1=HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Authentication\LogonUI
\SessionData"

SET
"regPath2=HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Authentication\LogonUI
\SessionData\1"

FOR /F "tokens=3" %%A IN ('REG QUERY "%regPath1%" /V LoggedOnUser /REG:64
2>nul ^| FIND "REG_SZ"') DO (SET "fullUser=%%A")

FOR /F "tokens=3" %%A IN ('REG QUERY "%regPath2%" /V LoggedOnUser /REG:64
2>nul ^| FIND "REG_SZ"') DO (SET "fullUser=%%A")

FOR /F "tokens=2 delims=\" %%B IN ("!fullUser!") DO (SET "UPN=%%B")

ECHO !UPN!

ENDLOCAL

ECHO. & PAUSE
```



## Method 2

---

**Scope:** Admin, System Account, PC, VDI

Returns the UPN. Requires a Microsoft Identity, like 365.

```
@ECHO OFF

SET
"regPath=HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Authentication\LogonUI\SessionData"

SET "regValue=LoggedOnUserSID"

:: Return User SID
FOR /F "tokens=* %%A IN ('REG QUERY "%regPath%" /S /REG:64 2^>nul ^| FINDSTR
/R /C:"HKEY_LOCAL_MACHINE") DO (
    FOR /F "tokens=3 %%B IN ('REG QUERY "%A" /V "%regValue%" /REG:64 2^>nul
    ^| FIND "REG_SZ") DO (
        SET "uSID=%%B"
    )
)

:: Returns the UPN by using the user SID from above.
FOR /F "tokens=3 %%C IN ('REG QUERY
"HKEY_USERS\%uSID%\Software\Microsoft\Office\16.0\Common\LanguageResources\Loca
lCache" /V "RegionalAndLanguageSettingsAccount" /REG:64') DO SET "UPN=%C"

ECHO %UPN%
ECHO. & PAUSE
```

## NOTE

---

### Other UPN Locations

HKLM\SOFTWARE\Microsoft\Enrollments\781CFF59-6FEF-44D0-B21D-111111111111 | UPN

HKU\User-SID-Here\Software\Microsoft\Office\16.0\Common\LanguageResources\LocalCache | RegionalAndLanguageSettingsAccount

HKU\User-SID-Here\Software\Microsoft\Office\16.0\Common\Identity | ADUsername

HKU\User-SID-Here\Software\Microsoft\Windows\CurrentVersion\Explorer\Substrate | The-UPN-Name:Substrate



## Method 3

**Scope:** Admin, System Account, PC, VDI

This is a more robust script for retrieving a UPN. It retrieves the user SID from the registry under SessionData, then attempts to find the User Principal Name (UPN) by querying multiple registry paths under HKEY\_USERS using the SID. It first checks for the UPN in specific locations related to Office and Remote Desktop, and if not found, continues checking other registry entries until the UPN is retrieved or an error message is displayed. If the UPN is found, it is returned; otherwise, the script informs the user that the UPN could not be retrieved.

```
@ECHO OFF

:: Define Variables
SET
"regPath=HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Authentication\LogonUI\SessionData"

SET "regKey=LoggedOnUserSID"

:: Return User SID
FOR /F "tokens=* %%A IN ('REG QUERY "%regPath%" /S /REG:64 2^>nul ^| FINDSTR
/R /C:"HKEY_LOCAL_MACHINE") DO (
    FOR /F "tokens=3 %%B IN ('REG QUERY "%A" /V "%regKey%" /REG:64 2^>nul ^|
    FIND "REG_SZ") DO (
        SET "uSID=%%B"
    )
)

:: Try to retrieve UPN from multiple locations.
SET "UPN="
IF DEFINED uSID (
    :: Check RegionalAndLanguageSettingsAccount.
    FOR /F "tokens=3 %%C IN ('REG QUERY
    "HKEY_USERS\%uSID%\Software\Microsoft\Office\16.0\Common\LanguageResources\Loca
    lCache" /V "RegionalAndLanguageSettingsAccount" /REG:64 2^>nul') DO SET
    "UPN=%C"

    IF NOT DEFINED UPN (
        FOR /F "tokens=7 delims=\" %%D IN ('REG QUERY
        "HKEY_USERS\%uSID%\Software\Microsoft\RdClientRadc\https://rdweb.wvd.microsoft.
        com/api/arm/feeddiscovery" /REG:64 2^>nul') DO SET "UPN=%D"
    )
)
```



```
IF NOT DEFINED UPN (
    FOR /F "tokens=7 delims=" %%E IN ('REG QUERY
"HKEY_USERS\%u$ID%\Software\Microsoft\RdClientRadc\https://rdweb.wvd.microsoft.
com/api/feeddiscovery/webfeeddiscovery.aspx" /REG:64 2^>nul') DO SET "UPN=%E"
)

IF NOT DEFINED UPN (
    FOR /F "tokens=3 delims=" %%F IN ('REG QUERY
"HKEY_USERS\%u$ID%\Software\Microsoft\Office\16.0\Common\Licensing\LicensingNex
t\LicenseIdToEmailMapping" /REG:64 2^>nul ^| FIND "REG_SZ"') DO SET "UPN=%F"
)
)

:: Output UPN or Error Message
IF DEFINED UPN (
    ECHO The UPN is: %UPN%
) ELSE (
    ECHO UPN could not be retrieved from any registry paths.
)

ECHO. & PAUSE
```

## Return User Profile Path

-----  
**FIND | FOR | REG**

**Scope:** Admin, System Account, PC, VDI

Retrieves the user SID of the currently selected user from the registry and then uses the SID to query the user's profile path.

```
@ECHO OFF

:: Return User SID
FOR /F "tokens=3" %%A IN ('REG QUERY
"HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Authentication\LogonUI" /V
"SelectedUserSID" /REG:64 2^>nul ^| FIND "REG_SZ"') DO (SET "u$ID=%A")

:: Return Profile Path
FOR /F "tokens=2*" %%B IN ('REG QUERY
"HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows
NT\CurrentVersion\ProfileList\%u$ID%" /V "ProfileImagePath" /REG:64') DO (SET
"userProfilePath=%C")
```



```
ECHO Profile Path: %userProfilePath%
```

```
ECHO. & PAUSE
```

## Detect if User is 'Elevated' as Admin

---

NET

**Scope:** Current User

Detects if the user is running a script as elevated. To test, double-click and run as a restricted user, and then right-click and run as admin. You should see the True and False at work, which is our Boolean logic. This will be important to some of your scripts that require elevation.

```
@ECHO OFF
```

```
NET SESSION >nul 2>&1
IF %ERRORLEVEL% EQU 0 (
    ECHO Elevated: True
) ELSE (
    ECHO Elevated: False
)
ECHO. & PAUSE
```

## Add User to Administrators Group

---

FIND | FINDSTR | FOR | REG

**Scope:** Admin, System Account, PC, VDI

Retrieves the SID of the currently logged-on user from the registry, uses it to find the user's UPN (User Principal Name) from Office-related settings, and then attempts to add the user to the local Administrators group. If the device is Azure AD joined, it adds the Azure AD account; otherwise, it adds the domain account.

```
@ECHO OFF
```

```
SET
"regPath=HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Authentication\LogonUI\SessionData"
```



```
SET "regKey=LoggedOnUserSID"

:: Return User SID
FOR /F "tokens=*" %%A IN ('REG QUERY "%regPath%" /S /REG:64 2^>nul ^| FINDSTR
/R /C:"HKEY_LOCAL_MACHINE") DO (
    FOR /F "tokens=3" %%B IN ('REG QUERY "%A" /V "%regKey%" /REG:64 2^>nul ^|
    FIND "REG_SZ") DO (
        SET "uSID=%%B"
    )
)

:: Returns the UPN by using the user SID from above.
FOR /F "tokens=3" %%C IN ('REG QUERY
"HKEY_USERS\%uSID%\Software\Microsoft\Office\16.0\Common\LanguageResources\Loca
lCache" /V "RegionalAndLanguageSettingsAccount" /REG:64') DO SET "UPN=%C"

:: Detect if device is enrolled.
DSREGCMD /STATUS | FINDSTR /I "AzureAdJoined" | FINDSTR "YES" >nul

:: If enrolled, add user to the group using their UPN. So magical.
IF %ERRORLEVEL% EQU 0 (
    ECHO Attempting Azure Add...
    NET LOCALGROUP Administrators /ADD "AzureAD\%UPN%" >nul 2>&1
) ELSE (
    ECHO Attempting Domain Add...
    NET LOCALGROUP Administrators /ADD "%UPN%" >nul 2>&1
)
ECHO. & ECHO Done!
ECHO. & PAUSE
```

## Return User Session Information

-----  
[ENABLEDELAYEDEXPANSION](#) | [FIND](#) | [FOR](#) | [QUSER](#) | [QWINSTA](#) | [REG](#)

The root of registry subkey LogonUI is meant for a single user, physical machine. If this is for a virtual environment, see Note at the bottom of code.

```
@ECHO OFF
SETLOCAL EnableDelayedExpansion
SET tstQUSER=NA
SET tstQWin=NA
SET quserSessId=NA

:: Return User SID
```



```
FOR /F "tokens=3" %%B IN ('REG QUERY
"HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Authentication\LogonUI" /V
"SelectedUserSID" /REG:64 2^>nul ^| FIND "REG_SZ"') DO (SET "uSID=%%B")

:: Return Username
FOR /F "tokens=3" %%C IN ('REG QUERY
"HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Authentication\LogonUI" /V
"LastLoggedOnUser" /REG:64 2^>nul ^| FIND "REG_SZ"') DO (SET "uName=%%C")

:: Let's verify quser.exe actually exists. If so, run logic for variables.
IF EXIST C:\Windows\System32\quser.exe (
    IF %ERRORLEVEL% EQU 0 (
        FOR /F "tokens=3" %%I IN ('QUSER 2^>nul') DO SET quserSessId=%%I
        SET "tstQUSER=TRUE"
        ECHO !quserSessId!
        ECHO.
    )
)

:: Let's verify qwinsta.exe actually exists. If so, run logic for variables.
IF EXIST C:\Windows\System32\qwinsta.exe (
    FOR /F "tokens=1-6" %%A IN ('QWINSTA 2^>nul') DO (
        IF "%B"=="!uName!" (
            SET "quinstaSessId=%C"
            ECHO QWINSTA: !uName! !quinstaSessId!
            SET "tstQWin=TRUE"
            ECHO.
        )
    )
)

CLS
ECHO QUSER: %tstQUSER%
ECHO QWINSTA: %tstQWin%
ECHO Username: %uName%
ECHO Session Id: %quserSessId%
ECHO Command: logoff %quserSessId%
ECHO.

ENDLOCAL

ECHO. & PAUSE
```



## NOTE

---

### Other Reg Keys

Virtual environments often require different handling for logged-on users. In virtual sessions, user data is typically stored in the `SessionData` and its subkeys, located deeper within the `LogonUI` registry hive. It's important to understand that virtual environments may not always align with the registry keys and file paths found on physical machines. This is why you need to test, test, test.

Checks if the username (`uName`) is ".\Administrator" and, if so, retrieves the logged-on SAM username from the registry path `SessionData\2`.

```
:: Get Username
IF "%uName%"==".\Administrator" (
    FOR /F "tokens=3" %%C IN ('REG QUERY
"HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Authentication\LogonUI\SessionD
ata\2" /V "LoggedOnSAMUser" /REG:64 2>nul ^| FIND "REG_SZ") DO (SET
"uName=%%C")
)
```

Similarly checks if the username (`uName`) is ".\Administrator" and, if so, retrieves the logged-on SAM username from the registry path `SessionData\1`.

```
IF "%uName%"==".\Administrator" (
    FOR /F "tokens=3" %%C IN ('REG QUERY
"HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Authentication\LogonUI\SessionD
ata\1" /V "LoggedOnSAMUser" /REG:64 2>nul ^| FIND "REG_SZ") DO (SET
"uName=%%C")
)
```



## Return User SIDs from HKEY\_USERS

**COLOR | ENABLEDELAYEDEXPANSION | FINDSTR | FOR | REG**

Detects the computer name and retrieves user SIDs from the HKEY\_USERS registry hive, filtering them based on specific patterns (S-1-5-21- and S-1-12-1-). It dynamically builds a file (SIDs.txt) containing the valid user SIDs and then outputs each SID stored in memory, pausing briefly between each one. The process includes cleaning up any existing SIDs.txt file before beginning the SID collection and export.

```
@ECHO OFF
TITLE HKEY_USERS SIDS DEMO
COLOR 0A
SETLOCAL EnableDelayedExpansion

SET "SIDs=SIDs.txt"

:: USER SID PATTERNS TO RETURN
:: Other patterns may need to be added.
SET "sidPattern1=S-1-5-21-"
SET "sidPattern2=S-1-12-1-"

CD "%~dp0"

IF EXIST SIDs.txt DEL /Q SIDs.txt

CLS
ECHO RETURN USER SIDS
ECHO.

:: DYNAMICALLY BUILD USER SID FILE FROM ONLINE VMs

ECHO Detecting computer name...
:: Why do I use ping, when I could just use timeout?
:: Try timeout in the system account. You'll see an issue.
PING -n 6 -w 1000 localhost >nul
ECHO Name: %COMPUTERNAME%
ECHO.

ECHO Generating User SIDs.txt data file...
PING -n 6 -w 1000 localhost >nul
IF EXIST SIDs.txt DEL /Q SIDs.txt

:: Return User SIDs
FOR /F "tokens=1,2,* delims= " %%S IN ('REG QUERY "HKEY_USERS" /REG:64 2^>nul
^| FINDSTR /R /C:"!sidPattern1!"') DO (
    ECHO %%S | FINDSTR /R /C:"_Classes" >nul
    IF ERRORLEVEL 1 (
        ECHO %%S
        SET "allSIDs=%%S !allSIDs!"
```



```
ECHO %%S >> SIDs.txt
)
)

ECHO.
ECHO User SIDs exported to SIDs.txt data file.
ECHO.
ECHO.
ECHO Returning each User SID stored in memory...
ECHO.

FOR %%A IN (%allSIDs%) DO (
    ECHO %%A
    PING -n 2 -w 1000 localhost >nul
)
ECHO.
ENDLOCAL

ECHO. & PAUSE
```

## Return Username, SID, User Profile & More

[DATE](#) | [DSREGCMD](#) | [ENABLEDELAYEDEXPANSION](#) | [FIND](#) | [FOR](#) | [REG](#) | [TIME](#) | [WMIC](#)

Collects local telemetry data about the current user, including the user name, SID, elevation status, FSLogix and AzureAD status, Chrome version, Windows build version, and a formatted timestamp of when the data was retrieved. It then outputs this data in a readable format before cleaning up and exiting. The script also checks for specific conditions like FSLogix and Azure AD presence and determines if the user has administrator privileges.

```
@ECHO OFF
TITLE Local Telemetry Data
COLOR 0A
SETLOCAL EnableDelayedExpansion

CD "%~dp0"
CLS

:: Define Variables
SET "elevationStatus=USER"
SET "uSID="
SET "fslogixPath=C:\Program Files\FSLogix"
SET "chromePath=C:\Program Files\Google\Chrome\Application\chrome.exe"

:: Detect FSLogix
IF EXIST "%fslogixPath%" (
```



```
        SET "FsLogix=TRUE"
) ELSE (
    SET "FsLogix=FALSE"
)

:: Detect AzureAD
FOR /F "tokens=2 delims=: %%A IN ('DSREGCMD /STATUS ^| FIND "AzureAdJoined"') DO (
    SET "azureAD=FALSE"
    IF "%%%A==" YES" SET "azureAD=TRUE"
)

:: Return Chrome Version
IF EXIST "%chromePath%" (
    FOR /F "tokens=* %%A IN ('WMIC datafile where
    "name='%ChromePath%:\=' get version /value ^| FIND "="') DO (
        FOR /F "tokens=2 delims==" %%B IN ("%%%A") DO SET
        "chromeVersion=%%%B"
    )
)

:: Return Windows Build Version
FOR /F "tokens=* %%A IN ('WMIC os get version /value ^| FIND "="') DO (
    FOR /F "tokens=2 delims==" %%B IN ("%%%A") DO SET "winVersion=%%%B"
)

:: Get Timestamp using the local time command.
FOR /F "tokens=1,2 delims= " %%A IN ('DATE /T') DO (
    SET "Today=%%%A"
)

FOR /F "tokens=1,2 delims= " %%A IN ('TIME /T') DO (
    SET "Today=%Today% %%A"
)

:: Return User SID
FOR /F "tokens=3 %%B IN ('REG QUERY
"HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Authentication\LogonUI" /V
"SelectedUserSID" /REG:64 2>nul ^| FIND "REG_SZ"') DO (SET "uSID=%%%B")

:: Return Username
FOR /F "tokens=3 %%C IN ('REG QUERY
"HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Authentication\LogonUI" /V
"LastLoggedOnUser" /REG:64 2>nul ^| FIND "REG_SZ"') DO (SET "uName=%%%C")

IF "%uName%"=="Administrator" (
```



```
FOR /F "tokens=3" %%C IN ('REG QUERY
"HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Authentication\LogonUI\SessionD
ata\2" /V "LoggedOnSAMUser" /REG:64 2>nul ^| FIND "REG_SZ") DO (SET
"uName=%%C")
)

IF "%uName%"=="\"Administrator" (
    FOR /F "tokens=3" %%C IN ('REG QUERY
"HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Authentication\LogonUI\SessionD
ata\1" /V "LoggedOnSAMUser" /REG:64 2>nul ^| FIND "REG_SZ") DO (SET
"uName=%%C")
)

:: Check Elevation Status
NET LOCALGROUP Administrators | FIND /I "%uName%" >nul 2>&1
IF NOT ERRORLEVEL 1 (
    SET "elevationStatus=ADMIN"
)

:: Get Current Date
FOR /F "tokens=2 delims= " %%A IN ('DATE /T') DO SET "currDate=%%A"

FOR /F "tokens=1-3 delims=/-. " %%A IN ("%currDate%") DO SET
"timestamp_date=%%C%%A%%B"

:: Get Current Time
FOR /F "tokens=1-2 delims=: " %%A IN ('TIME /T') DO SET "currTime=%%A%%B"

FOR /F "tokens=1-2 delims= " %%A IN ("%currTime%") DO SET "timestamp_time=%%A"
rem %%B

:: Create the formatted timestamp.
SET "timestamp=%timestamp_date%_%timestamp_time%"

:: Output Telemetry
ECHO LOCAL TELEMETRY DATA
ECHO USER: %uName%
ECHO ELEVATION: %elevationStatus%
ECHO FSLOGIX: %FsLogix%
ECHO AZUREAD: %azureAD%
ECHO PROFILE: %USERPROFILE%
ECHO USER SID: %uSID%
ECHO WIN BUILD: %WinVersion%
ECHO CHROME: %ChromeVersion%
ECHO TIMESTAMP: %timestamp%
ECHO.
```



```
:: Cleanup and Exit  
:: Do stuff here  
ENDLOCAL
```

```
ECHO.  
PAUSE
```

```
EXIT /B 0
```



## Batch: File & Folder Operations

---

How to test from the System Account ([PsExec](#)):

```
PsExec64 -S CMD  
"C:\Batch\script.cmd"
```

### Copy File

---

**COPY**

```
:: Compact Code  
SET "src=C:\Batch\source.txt" & SET "dst=C:\Batch\copy\source.txt"  
COPY /Y "%src%" "%dst%" & PAUSE
```

Or

```
:: -----  
:: LAB SETUP  
MD C:\Batch  
ECHO. >"C:\Batch\source.txt"  
CLS  
:: -----  
  
@ECHO OFF  
  
SETLOCAL  
  
:: Set source and destination paths.  
SET "sourceFile=C:\Batch\source.txt"  
SET "destinationFile=C:\Batch\copy\destination.txt"  
  
:: Copy the file to the destination.  
COPY /Y "%sourceFile%" "%destinationFile%" >nul  
IF %ERRORLEVEL% EQU 0 (  
    ECHO File copied successfully to: %destinationFile%  
) ELSE (  
    ECHO Failed to copy the file.  
)  
ENDLOCAL  
ECHO. & PAUSE
```



## Xcopy, Files & Folders

### XCOPY

```
:: Compact Code
SET "src=C:\Batch\source.txt" & SET "dst=C:\Batch\copy\source.txt"

:: For Files
XCOPY /Y /Q /C "%src%" "%dst%\"
:: Or
ECHO F|XCOPY /Y /Q /C "%src%" "%dst%"

:: For Folders
XCOPY "C:\Batch\copy1" "C:\Batch\copy2" /E /H /K /Y
PAUSE
```

## Robocopy, Files & Folders

### ROBOCOPY

**Robocopy**, or "Robust File Copy," is a command-line tool in Windows that is designed for efficiently copying and moving files and directories. It provides more powerful and reliable file-copying capabilities than the basic copy or xcopy commands. **Robocopy** is especially useful when dealing with large datasets, network transfers, or situations where you need to copy files in a robust and fault-tolerant manner.

### Copy File to Folder

```
:: Compact Code
SET "src=C:\Batch" & SET "dst=C:\Batch\copy"
ROBOCOPY "%src%" "%dst%" "source.txt" /NFL /NDL /NJH /NJS /NP /XO
ECHO. & PAUSE
```

Or

```
:: -----
:: LAB SETUP
MD C:\Batch & MD C:\Batch\copy
ECHO This is some text. > "C:\Batch\DataFile.txt"
CLS
:: -----
```



```
@ECHO OFF

SETLOCAL

:: Set source and destination paths.
SET "Filename=DataFile.txt"

SET "sourcePath=C:\Batch"
SET "destinationPath=C:\Batch\copy"

:: Display the paths for confirmation.
ECHO Filename: %Filename%
ECHO Source Folder: %sourcePath%
ECHO Destination Folder: %destinationPath%

:: Use ROBOCOPY to copy the file without copying auditing info.
ROBOCOPY "%sourcePath%" "%destinationPath%" "%Filename%" /NFL /NDL /NJH /NJS
/COPY:D

:: Check for error level and confirm the result.
IF %ERRORLEVEL% LSS 8 (
    ECHO File copied successfully to: %destinationPath%\%Filename%
) ELSE (
    ECHO Failed to copy the file. Error Level: %ERRORLEVEL%
)

ENDLOCAL

ECHO. & PAUSE
```

## Copy Folder to Folder, Keep Source

---

```
:: Folder to Folder. Keep Source.
@ECHO OFF

SET "src=C:\Batch\copy1"
SET "dst=C:\Batch\copy2"

ROBOCOPY "%src%" "%dst%" /E /NFL /NDL /NJH /NJS /NP

ECHO.
PAUSE
```



## Copy Folder to Folder, Delete Source

---

```
:: Folder to Folder. Delete Source Dangerzone
@ECHO OFF

SET "src=C:\Batch\copy1" & SET "dst=C:\Batch\copy2"

ROBOCOPY "%src%" "%dst%" /MOVE /E /NFL /NDL /NJH /NJS /NP
:: Ensure files are being copied.
:: ROBOCOPY "%src%" "%dst%" /E /IS /IT /NFL /NDL /NJH /NJS /NP

ECHO. & PAUSE
```

## Delete File

---

### DEL

```
:: Compact Code
SET "src=C:\Batch\source.txt"
DEL /Q "%src%" & PAUSE
```

### Or

```
:: -----
:: LAB SETUP
MD C:\Batch
ECHO. >"C:\Batch\file-to-delete.txt"
CLS
:: -----

@ECHO OFF

SETLOCAL

:: Set File Path
SET "filePath=C:\Batch\file-to-delete.txt"

:: Delete File
IF EXIST "%filePath%" (
    DEL /Q "%filePath%" >nul
    IF NOT EXIST "%filePath%" (
        ECHO File deleted successfully: %filePath%
    ) ELSE (

```



```
ECHO Failed to delete the file.  
 )  
) ELSE (   
 ECHO %filePath% does not exist.  
)  
ENDLOCAL  
  
ECHO. & PAUSE
```

## Delete Folder

### RD | ROBOCOPY

```
:: Compact Code  
SET "src=C:\Batch\NewFolder"  
RD /Q "%src%" & PAUSE  
:: /Q /S - Dangerzone. Be careful.
```

Or

```
SET "src=C:\Batch\NewFolder"  
ROBOCOPY %src% %src% /S /MOVE  
PAUSE
```

## Set Current Working Directory

### PUSHD | POPD

```
@ECHO OFF  
  
:: This works great for setting the current folder.  
:: But also see pushd and popd.  
CD "%~dp0"  
ECHO %CD%  
  
ECHO.  
PAUSE
```



## Return File LastWriteTime

-----  
**FOR**

**LastWriteTime** is a file system property that indicates the most recent date and time when the content of a file was modified. Can be used with conditional logic to perform operations based on **LastWriteTime**.

```
@ECHO OFF

SETLOCAL EnableDelayedExpansion

:: Get LastWriteTime from the file using PowerShell.
FOR /F "delims=%" %%D IN ('powershell -Command "(Get-Item
'C:\Windows\notepad.exe').LastWriteTime.ToString('yyyyMMdd'))'') DO SET X=%%D

:: Output the formatted date.
ECHO Modified Date: !X!

ENDLOCAL & ECHO. & PAUSE
```

## Return File CreationTime

-----  
**FOR**

**CreationTime** is a file system property that represents the date and time when a file was originally created on the file system. Can be used with conditional logic to perform operations based on **CreationTime**.

```
@ECHO OFF

SETLOCAL EnableDelayedExpansion

:: Get CreationTime from the file using PowerShell.
FOR /F "delims=%" %%D IN ('powershell -Command "(Get-Item
'C:\Windows\system32\notepad.exe').CreationTime.ToString('yyyyMMdd'))'') DO SET
Y=%%D

:: Output the formatted date.
ECHO Creation Date: !Y!

ECHO. & PAUSE
```



## File LastWriteTime/CreationTime, 30 Days

---

Check if the returned date attributes—**LastWriteTime/CreationTime**—is older than 30 days. Change the conditional logic to meet your requirements, and add your own code to perform actions based on those two attributes. Could prove to be quite useful.

```
@ECHO OFF
SETLOCAL EnableDelayedExpansion

SET "filePath=C:\Batch\Test.txt"
SET checkDays=30
SET "msg=Older than %checkDays% days"

:: Get the current date once.
FOR /F "delims=%" %%G IN ('powershell -Command
"(Get-Date).ToString('yyyyMMdd')")' DO SET CURRENT_DATE=%%G

:: Get LastWriteTime and CreationTime from the file in one PowerShell call.
FOR /F "delims=%" %%F IN ('powershell -Command "(Get-Item '%filePath%') |
Select-Object LastWriteTime, CreationTime | ForEach-Object {
$_.LastWriteTime.ToString('yyyyMMdd') + ' ' +
$_.CreationTime.ToString('yyyyMMdd') }")' DO (
    SET X=%%F
)

:: Parse the dates from the combined output.
FOR /F "tokens=1,2" %%H IN ("!X!") DO (
    SET LastWriteTime=%%H
    SET CreationTime=%%I
)

:: Check LastWriteTime
SET /A DIFF_X=(CURRENT_DATE - %LastWriteTime%)
ECHO LastWriteTime: %LastWriteTime%
IF %DIFF_X% GTR %checkDays% (ECHO %msg%: TRUE) ELSE (ECHO %msg%: FALSE)

ECHO.

:: Check CreationTime
SET /A DIFF_Y=(CURRENT_DATE - %CreationTime%)
ECHO CreationTime: %CreationTime%
IF %DIFF_Y% GTR %checkDays% (ECHO %msg%: TRUE) ELSE (ECHO %msg%: FALSE)
ENDLOCAL

ECHO. & PAUSE
```



## Copy File to Each User Profile

```
COPY | ENABLEDELAYEDEXPANSION | FOR

:: -----
:: LAB SETUP
MD C:\Batch
ECHO. >"C:\Batch\source.txt"
CLS
:: -----

@ECHO OFF

SETLOCAL EnableDelayedExpansion

:: Define Source File
SET "sourceFile=C:\Batch\source.txt"

:: Loop through each folder in C:\Users.
FOR /D %%U IN (C:\Users\*) DO (
    IF /I NOT "%U%"=="C:\Users\DefaultAppPool" IF /I NOT
    "%U%"=="C:\Users\Public" (
        SET "destPath=%U\Desktop"
        ECHO Copying "%sourceFile%" to "!destPath!"
        COPY /Y "%sourceFile%" "!destPath!" >nul
    )
)
ENDLOCAL

ECHO. & ECHO Done!
ECHO. & PAUSE
```



## Delete File from Each User Profile

-----  
**DEL | ENABLEDELAYEDEXPANSION | FOR**

```
:: -----
:: LAB SETUP
ECHO. >"%USERPROFILE%\Desktop\source.txt"
CLS
:: -----

@ECHO OFF

SETLOCAL EnableDelayedExpansion

:: Define File Name
SET "fileToDelete=source.txt"

:: Loop through each folder in C:\Users.
FOR /D %%U IN (C:\Users\*) DO (
    IF /I NOT "%U%"=="C:\Users\DefaultAppPool" IF /I NOT
    "%U%"=="C:\Users\Public" (
        SET "destPath=%U\Desktop"

        :: Check if the file exists before attempting to delete.
        IF EXIST "!destPath!\%fileToDelete%" (
            ECHO Deleting "!destPath!\%fileToDelete%"
            DEL /F /Q "!destPath!\%fileToDelete%"
        ) ELSE (
            ECHO File not found in "!destPath!"
        )
    )
)

ENDLOCAL

ECHO .
ECHO Done!

ECHO .
PAUSE
```



## Read Contents of File

-----  
**ENABLEDELAYEDEXPANSION | FOR**

```
:: -----
:: LAB SETUP
MD C:\Batch
(
    ECHO Line1 & ECHO Line2 & ECHO Line3
) > "C:\Batch\source.txt"
CLS
:: -----

@ECHO OFF

SETLOCAL EnableDelayedExpansion

:: Set File Path
SET "filePath=C:\Batch\source.txt"

:: Initialize the fileContent variable to hold the content.
SET "Content1="
SET "Content2="

:: Read the file line by line and append to the variable.
FOR /F "delims=" %%A IN ('TYPE "%filePath%"') DO (
    SET "Content1=!Content1! %%A"
    SET "Content2= %%A!Content2!"
)
ECHO File Contents:
ECHO !Content1!
ECHO !Content2!

ENDLOCAL

ECHO .
ECHO Done!

ECHO .
PAUSE
```



## Check if Folder is Empty

-----  
**DEL | ENABLEDELAYEDEXPANSION | FOR**

```
:: Compact Code
@ECHO OFF
SET "EMPTY=NA
FOR /F "delims=%" %%A IN ('DIR /B "C:\Batch" 2>nul') DO (SET "EMPTY=FALSE" &
GOTO :END) & SET "EMPTY=TRUE"
:END
ECHO %EMPTY% & PAUSE
```

Or

```
@ECHO OFF
SETLOCAL EnableDelayedExpansion

:: Path to Check. Here we are checking Firefox profiles.
SET "FolderPath=%localappdata%\Mozilla\Firefox\Profiles"

:: Initialize EMPTY as TRUE
SET "EMPTY=TRUE"

:: Test if the profiles directory contains files or subdirectories.
ECHO Testing %FolderPath%...

FOR /F %%I IN ('DIR /B /A "%FolderPath%" 2>nul') DO (
    SET "EMPTY=FALSE"
    GOTO :CHECK
)

:CHECK
ECHO Profiles Empty: %EMPTY%
ECHO.

:: Perform actions if EMPTY is TRUE.
IF "%EMPTY%"=="TRUE" (
    ECHO Deleting profiles.ini and installs.ini...
    DEL /Q "%FolderPath%\profiles.ini" 2>nul
    DEL /Q "%FolderPath%\installs.ini" 2>nul
) ELSE (
    ECHO Profiles directory is not empty.
)
ENDLOCAL
ECHO. & ECHO Done! & ECHO. & PAUSE
```



## Delete Folders Older Than 30 Days

**ENABLEDELAYEDEXPANSION | FORFILES | PUSHD | POPD | RD | TIMEOUT**

Delete profile folders older than 30 days. The evaluated days can be changed using SET Days. Note, these use the 'date created' property for the conditional logic.

```
@ECHO OFF
SETLOCAL EnableDelayedExpansion

:: Define Variables
SET "folderPath=%localappdata%\Mozilla\Firefox\Profiles"
SET Days=-30

IF EXIST "%folderPath%" (
    ECHO Deleting profiles from %folderPath%...
    TIMEOUT /T 3 >nul
    CD "%folderPath%"
    FORFILES /D %Days% /C "CMD /C IF @isDIR == TRUE RD /S /Q @path"
    ECHO .
)
ECHO. & ECHO Done! & ECHO. & PAUSE
```

Or

```
@ECHO OFF

SETLOCAL EnableDelayedExpansion

:: Define Variables
SET "folderPath=%localappdata%\Mozilla\Firefox\Profiles"
SET Days=-30

:: Check if the Profiles directory exists.
IF EXIST "%folderPath%" (
    ECHO Deleting profiles from %folderPath%...
    TIMEOUT /T 3 >nul

    :: Change to the target directory safely.
    PUSHD "%folderPath%" ||
        ECHO Failed to access directory. Exiting. & TIMEOUT /T 3 >nul
        EXIT /B 1
    )
    :: Remove directories older than 30 days.
    FORFILES /D %Days% /C "CMD /C IF @isDIR == TRUE RD /S /Q @path"
```



```
    :: Return to the original directory.  
    POPD  
    ECHO.  
 ) ELSE (   
    ECHO No Profiles directory found at %FolderPath%.  
 )  
  
ECHO. & ECHO Done! & ECHO. & PAUSE
```

## Return MD5 for File

**CERTUTIL | ENABLEDELAYEDEXPANSION | FINDSTR | FOR**

```
@ECHO OFF  
  
:: Retrieve the MD5 hash.  
FOR /F "tokens=1 delims=" %%A IN ('CERTUTIL -HASHFILE  
"C:\Windows\system32\notepad.exe" MD5 ^| FINDSTR /I /V "MD5 hash of file"') DO  
SET "Hash=%%A"  
  
ECHO Our file: C:\Windows\system32\notepad.exe  
ECHO MD5: %Hash%  
  
ECHO. & ECHO Done! & ECHO. & PAUSE
```

## Verify MD5 for File

**CERTUTIL | ENABLEDELAYEDEXPANSION | FINDSTR | FOR**

```
@ECHO OFF  
SETLOCAL EnableDelayedExpansion  
  
:: Retrieve the MD5 hash.  
FOR /F "tokens=1 delims=" %%A IN ('CERTUTIL -HASHFILE  
"C:\Windows\system32\notepad.exe" MD5 ^| FINDSTR /I /V "MD5 hash of file"') DO  
SET Hash=%%A  
  
ECHO Our file: C:\Windows\system32\notepad.exe & TIMEOUT /T 2 >nul  
ECHO MD5: !Hash! & TIMEOUT /T 2 >nul  
ECHO Let's change the MD5 stored in memory.  
SET Hash=a96a626f8b6592ad7b3beb5ae61e3a99 & TIMEOUT /T 2 >nul
```



```
ECHO MD5: a96a626f8b6592ad7b3beb5ae61e3a99 & TIMEOUT /T 2 >nul
ECHO Let's compare the updated MD5 with notepad.exe... & TIMEOUT /T 2 >nul
:: Compare the hash.
FOR /F "tokens=1 delims=" %%B IN ('CERTUTIL -HASHFILE
"C:\Windows\system32\notepad.exe" MD5 ^| FINDSTR /I /V "MD5 hash of file"') DO
(
    IF %%B==!Hash! (ECHO TRUE. MD5 matches.) ELSE (ECHO FALSE. MD5 does not
    match.)
) & TIMEOUT /T 2 >nul

ENDLOCAL

ECHO. & ECHO Done!
ECHO. & PAUSE
```

## Return MD5s for Temp Folder Files

---

[CERTUTIL](#) | [ENABLEDELAYEDEXPANSION](#) | [FINDSTR](#)

```
@ECHO OFF
SETLOCAL EnableDelayedExpansion

:: Return MD5 hashes of files in %TEMP%.
SET "Directory_Path=%TEMP%"

:: Process each file in the temp folder.
FOR %%F IN ("%Directory_Path%\*") DO (
    FOR /F "delims=" %%A IN ('CERTUTIL -HASHFILE "%F" MD5 ^| FINDSTR /I /V
    "MD5 hash of file" ^| FINDSTR /R /V "^\$"') DO (
        ECHO %%~NFX %%A
        ECHO %%~NFX,%%A >> log.txt
    )
)

ENDLOCAL
ECHO. & ECHO Done! & ECHO. & PAUSE
```

## NOTE

---

An MD5 hash is a unique fixed-length string (a 128-bit hexadecimal value) that represents the contents of a file. It is used to verify file integrity—if a file is altered, its MD5 hash will change.



## Set Permissions on Files or Folders

### CACLS | ICACLS

```
@ECHO OFF

:: Set permissions on a file.
CACLS "C:\Batch\file.txt" /E /G Everyone:F
CACLS "C:\Batch\file.txt" /E /G The-User-Name:R
CACLS "C:\Batch\file.txt" /E /G "Power Users":C

:: Set permissions on a folder.
CACLS "C:\Batch\Folder" /T /E /G Everyone:F
```

Or

```
:: Set permissions on a file.
ICACLS "C:\Batch\file.txt" /GRANT Everyone:(F)
ICACLS "C:\Batch\file.txt" /GRANT The-User-Name:(R)
ICACLS "C:\Batch\file.txt" /DENY The-User-Name:(F)
ICACLS "C:\Batch\file.txt" /GRANT "Power Users":(M)
ICACLS "C:\Batch\file.txt" /SETOWNER The-User-Name
ICACLS "C:\Batch\file.txt" /REMOVE Everyone

:: Set permissions on a folder.
ICACLS "C:\Batch\Folder" /GRANT Everyone:(F) /T /C
ICACLS "C:\Batch\Folder" /REMOVE Everyone /T

ECHO. & PAUSE
```

## Take Ownership of Files or Folders

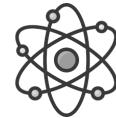
### TAKOWN

```
@ECHO OFF

:: Take ownership of a file.
TAKOWN /F "C:\Batch\file.txt"

:: Take ownership of a folder.
TAKOWN /F "C:\Batch\Folder" /R /D y

ECHO. & PAUSE
```



## Unblock Downloaded File

### UNBLOCKING FILES

When you download files from the internet, sometimes the files are automatically identified as a security risk which prompts a pop-up (the files have a block flag). This is how you prevent the extra pop-up.

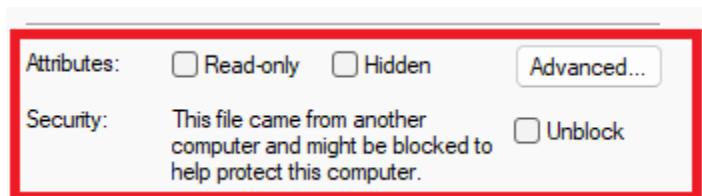
```
:: -----
:: LAB SETUP
MD C:\Batch
ECHO Test Data >C:\Batch\source.txt"
CLS
:: -----
@ECHO OFF

ECHO.>C:\Batch\source.txt:Zone.Identifier

ECHO.
ECHO Done!

ECHO.
PAUSE
```

If you right-click on a file and go to properties, you can view the security status. Look at the *Unblock*. That is what triggers the extra security prompt. Click it to disable the extra pop up.





## Batch: Text File Operations

---

How to test from the System Account ([PsExec](#)):

```
PsExec64 -S CMD  
"C:\Batch\script.cmd"
```

### Create Text File

---

```
-----  
TIMEOUT  
  
:: -----  
:: LAB SETUP  
MD C:\Batch  
CLS  
:: -----  
  
@ECHO OFF  
  
SET "filePath=C:\Batch\File.txt"  
  
ECHO. >"%filePath%"  
  
IF EXIST "%filePath%" ECHO File Created! & TIMEOUT /T 2 >nul  
  
ECHO. & ECHO Done!  
ECHO. & PAUSE
```

### Write to Text File

---

```
-----  
:: -----  
:: LAB SETUP  
MD C:\Batch  
CLS  
:: -----  
  
@ECHO OFF  
SET "filePath=C:\Batch\File.txt"
```



```
:: Writing to Text File (Overwrites)
ECHO %USERNAM% > "%filePath%"

ECHO. & ECHO Done!
ECHO. & PAUSE
```

## Append to Text File

---

### TIMEOUT

```
:: -----
:: LAB SETUP
MD C:\Batch
ECHO Hello, %USERNAME%! >"C:\Batch\File.txt"
CLS
:: -----

@ECHO OFF
SET "filePath=C:\Batch\File.txt"

ECHO Before: & TIMEOUT /T 2 >nul
TYPE "%filePath%"

ECHO.
ECHO Adding a new line... & ECHO. & TIMEOUT /T 2 >nul
ECHO %COMPUTERNAME% >> "%filePath%"

ECHO After: & TIMEOUT /T 2 >nul
TYPE "%filePath%"
ECHO. & ECHO Done! & ECHO. & TIMEOUT /T 2 >nul
ECHO. & PAUSE
```

## Read from Text File

---

### FOR

```
:: Compact
SETLOCAL EnableDelayedExpansion
FOR /F "delims=" %%A IN ("%filePath%") DO (SET "Content=!Content!%%A ")
ECHO %Content% & PAUSE
```



Or

```
:: -----
:: LAB SETUP
MD C:\Batch
ECHO Hello, %USERNAME%! >"C:\Batch\file.txt"
CLS
:: -----

@ECHO OFF
SETLOCAL EnableDelayedExpansion
SET "filePath=C:\Batch\file.txt"
:: Read Text File
SET "Content="

FOR /F "delims==" %%A IN (%filePath%) DO (
    SET "Content=!Content!%%A "
)

ECHO The content of the file is:
ECHO %Content%
ENDLOCAL

ECHO. & PAUSE
```

## Read from Text File, Line by Line

### ENABLEDELAYEDEXPANSION | FOR | TIMEOUT | TYPE

```
:: -----
:: LAB SETUP
MD C:\Batch
(
    ECHO Eddie Jackson
    ECHO Computer-Lab1
    ECHO TYNEKJG-12
) > "C:\Batch\file.txt"
CLS
:: -----

@ECHO OFF
SETLOCAL EnableDelayedExpansion
SET "filePath=C:\Batch\file.txt"
SET Count=1
```



```
:: Output Text File
ECHO The content of the file is:
TYPE "%filePath%" & ECHO. & TIMEOUT /T 2 >nul

ECHO Reading file line by line... & TIMEOUT /T 2 >nul
ECHO.

FOR /F "delims=%" %%A IN (%filePath%) DO (
    ECHO Line !Count!: %%A & TIMEOUT /T 2 >nul
    SET /A Count+=1
)
ENDLOCAL
ECHO. & PAUSE
```

## Read from Text File, Type.exe

### ENABLEDELAYEDEXPANSION | FOR

```
:: -----
:: LAB SETUP
MD C:\Batch
ECHO Hello, %USERNAME%! >"C:\Batch\File.txt"
CLS
:: -----

@ECHO OFF
SETLOCAL EnableDelayedExpansion

:: Define File Path
SET "filePath=C:\Batch\File.txt"
SET "Content="

:: Use the TYPE command and store its output in a variable.
FOR /F "delims=%" %%A IN ('TYPE "%filePath%"') DO (
    SET "Content=!Content!%%A "
)

:: Display Content
ECHO The content of the file is:
ECHO !Content!

ENDLOCAL
ECHO. & PAUSE
```



## Search & Replace String in Text File

[ENABLEDELAYEDEXPANSION](#) | [FIND](#) | [FOR](#) | [MOVE](#) | [TYPE](#)

```
:: -----
:: LAB SETUP
MD C:\Batch
(     ECHO Line 1
    ECHO 9F1AE7B34381DFD1111
    ECHO Line 3) > "C:\Batch\Text.dat"
CLS
:: -----

@ECHO OFF
SETLOCAL EnableDelayedExpansion

:: Define Variables
SET "Input=C:\Batch\Text.tmp"
SET "Output=C:\Batch\Text.dat"
SET "strOld=9F1AE7B34381DFD1111" &:: needs to be replaced
SET "strNew=9F1AE7B34381DFD9999" &:: replacement string

ECHO Content of file before:
TYPE "%Output%" & ECHO.

:: Prep Temp File
MOVE "%Output%" "%Input%" >nul
IF EXIST "%Output%" DEL /Q "%Output%"

:: Perform Search
FOR /F "tokens=1* delims=[]" %%A IN ('TYPE "%Input%" ^| FIND /N /V "")' DO (
    SET "strLine=%%B"
    :: Replace String If Found
    IF DEFINED strLine SET strLine=!strLine:%strOld%=%strNew%!
    >> "%Output%" ECHO.!strLine! 2>nul
)

ECHO Text.dat updated^^! & ECHO.

ECHO Content of file after:
TYPE "%Output%" & ECHO.

:: Cleanup
DEL /Q %Input%
SET %Output%=
SET %Input%=
SET %strOld%=
SET %strNew%=
ENDLOCAL

ECHO. & PAUSE
EXIT /B 0
```



## Sort Text File Ascending/Descending

**SORT | TIMEOUT | TYPE**

```
:: Compact Code
SORT "C:\Batch\Unsorted.txt" > "C:\Batch\Sorted.txt"
PAUSE
```

**Or**

```
:: -----
:: LAB SETUP
@ECHO OFF
MD C:\Batch

:: Unsorted.txt
(
    ECHO B
    ECHO D
    ECHO A
    ECHO F
    ECHO E
    ECHO C
) > "C:\Batch\Unsorted.txt"
CLS
:: -----

@ECHO OFF

SET "Unsorted=C:\Batch\Unsorted.txt"
SET "Sorted=C:\Batch\Sorted.txt"

CLS

:: Show Unsorted
ECHO Unsorted Content: & CALL :TIMER 2
TYPE "%Unsorted%"
ECHO.

:: Sort into Sorted.txt, ascending order.
ECHO Running our SORT command... & CALL :TIMER 2
SORT "%Unsorted%" > "%Sorted%"
:: Or Descending: SORT /R "C:\Batch\Unsorted.txt" > "C:\Batch\Sorted.txt"
ECHO.
```



```
:: Show Sorted

ECHO Sorted Content: & CALL :TIMER 2
TYPE "%Sorted%"

ECHO.
ECHO Done!

ECHO.
PAUSE

EXIT

:TIMER
:: This is our timer function.
:: You can use the CALL command to improve
:: code aesthetics, processing, and output.
:: You can call other scripts and labels.
:: Notice how we use seconds.

SET "Seconds=%~1"
ECHO.
TIMEOUT /T %Seconds% >nul
:: Good to use in the system account: ping localhost >nul
GOTO :EOF
```



## Merge Two Text Files

**CALL | TIMEOUT | TYPE**

```
:: Compact Code
SET "File1=C:\Batch\file1.txt" & SET "File2=C:\Batch\file2.txt"
SET "Combined=C:\Batch\Combined.txt"
COPY /B "%File1%" + "%File2%" "%Combined%" >nul
PAUSE
```

Or

```
:: -----
:: LAB SETUP
MD C:\Batch

:: File1.txt
(
    ECHO Line1
    ECHO Line2
    ECHO Line3
) > "C:\Batch\file1.txt"

:: File2.txt
(
    ECHO Line4
    ECHO Line5
    ECHO Line6
) > "C:\Batch\file2.txt"
CLS
:: -----

@ECHO OFF

:: Define Variables
SET "File1=C:\Batch\file1.txt"
SET "File2=C:\Batch\file2.txt"
SET "Combined=C:\Batch\Combined.txt"

ECHO Content of File1:
TYPE "%File1%" & CALL :TIMER 2

ECHO Content of File2:
TYPE "%File2%" & CALL :TIMER 2

ECHO Now let's merge files into Combined.txt... & CALL :TIMER 2
```



```
:: Merge File1.txt and File2.txt into Combined.txt.  
COPY /B "%File1%" + "%File2%" "%Combined%" >nul  
  
:: Confirmation  
ECHO Files have been merged! & CALL :TIMER 2  
  
ECHO Showing content of %Combined%... & CALL :TIMER 2  
TYPE "%Combined%" & CALL :TIMER 2  
  
ECHO. & ECHO Done!  
  
ECHO. & PAUSE  
  
:TIMER  
:: This is our timer function.  
:: You can use the CALL command to improve  
:: code aesthetics, processing, and output.  
:: You can call other scripts and labels.  
:: Notice how we use seconds.  
  
SET "Seconds=%~1"  
ECHO.  
TIMEOUT /T %Seconds% >nul  
:: Good to use in the system account: ping localhost >nul  
GOTO :EOF
```



## Merge Two Text Files, Remove Dups

**CALL | FOR | MOVE | TIMEOUT | TYPE**

```
:: -----

:: LAB SETUP
MD C:\Batch

:: File1.txt
(
    ECHO Line1
    ECHO Line2
    ECHO Line3
) > "C:\Batch\file1.txt"

:: File2.txt
(
    ECHO Line4
    ECHO Line2
    ECHO Line5
    ECHO Line6
) > "C:\Batch\file2.txt"
CLS
:: -----

@ECHO OFF

:: Define Variables
SET "File1=C:\Batch\file1.txt"
SET "File2=C:\Batch\file2.txt"
SET "Combined=C:\Batch\Combined.txt"

ECHO Content of File1:
TYPE "%File1%" & CALL :TIMER 2

ECHO Content of File2:
TYPE "%File2%" & CALL :TIMER 2

ECHO Now let's merge files into Combined.txt... & CALL :TIMER 2

:: Merge File1.txt and File2.txt into Combined.txt.
COPY /B "%File1%" + "%File2%" "%Combined%" >nul

:: Confirmation
ECHO Files have been merged! & CALL :TIMER 2
```



```
ECHO Showing content of %Combined%... & CALL :TIMER 2

TYPE "%Combined%" & CALL :TIMER 2

ECHO There are duplicate lines. & CALL :TIMER 4
ECHO We want to remove any duplicates. & CALL :TIMER 2
ECHO Removing dupes now... & CALL :TIMER 2

SET "tempFile=temp.txt"
SET "outputFile=removedDuplicates.txt"
ECHO.>%outputFile%

:: Remove Duplicates
FOR /F "usebackq delims=%" %%A IN ("%Combined%") DO (
    FINDSTR /X /C:"%%A" "%outputFile%" 2>nul || ECHO %%A>>"%outputFile%"
) >nul

:: Sort Ascending
SORT "%outputFile%" /O "tmp" & MOVE /Y "tmp" "%outputFile%" >nul

:: Delete Empty Lines
FOR /F "usebackq delims=%" %%B IN ("%outputFile%") DO (
    IF NOT "%%B""==" ECHO %%B>>"tmp"
)

MOVE /Y "tmp" "%outputFile%" >nul

ECHO Dupes have been removed! & CALL :TIMER 4
ECHO Let's verify... & CALL :TIMER 2
TYPE "%outputFile%" & CALL :TIMER 2

ECHO. & ECHO Done!

ECHO. & PAUSE

:TIMER
:: This is our timer function.
:: You can use the CALL command to improve
:: code aesthetics, processing, and output.
:: You can call other scripts and labels.
:: Notice how we use seconds.

SET "Seconds=%~1"
ECHO.
TIMEOUT /T %Seconds% >nul
:: Good to use in the system account: ping localhost >nul
GOTO :EOF
```



## Return Specific Line in Text File

**CALL | ENABLEDELAYEDEXPANSION | FOR | TIMEOUT | TYPE**

Let's say you have a config file with a hundred lines of settings. On line 50 is a setting you need as part of a custom script. This demonstrates how to acquire the contents of a specific line, line 2 in our case, but this could easily be line 50, or even line 500.

```
:: -----
:: LAB SETUP
MD C:\Batch
(
    ECHO %USERNAME%
    ECHO 9F1AE7B34381DFD1111
    ECHO %COMPUTERNAME%) > "C:\Batch\Text.dat"
CLS
:: -----

@ECHO OFF
SETLOCAL EnableDelayedExpansion
SET count=1
SET "filePath=C:\Batch\Text.dat"
SET strReturn=

CLS
ECHO Showing each line in %filePath%... & CALL :TIMER 2

:: Read our text file.
FOR /F "delims=" %%A IN ('TYPE "%filePath%"') DO (
    :: The conditional logic we use for the line we want.
    IF !count! EQU 2 SET "strReturn=%%A"
    ECHO !count! %%A & CALL :TIMER 2
    SET /A count+=1
)
ECHO.

ECHO Showing a specific line: & CALL :TIMER 2
ECHO Line 2: !strReturn! & CALL :TIMER 2
ENDLOCAL

ECHO. & ECHO Done!

ECHO. & PAUSE

:TIMER
:: This is our timer function.
:: You can use the CALL command to improve
```



```
:: code aesthetics, processing, and output.  
:: You can call other scripts and labels.  
:: Notice how we use seconds.  
  
SET "Seconds=%~1"  
TIMEOUT /T %Seconds% >nul  
:: Good to use in the system account: ping localhost >nul  
GOTO :EOF
```



## Batch: Process Operations

---

How to test from the System Account ([PsExec](#)):

```
PsExec64 -S CMD  
"C:\Batch\script.cmd"
```

### Start Process

---

#### START

```
:: Launch Notepad and wait until closed.  
START /WAIT Notepad.exe &:: /WAIT, /MIN, /MAX, /B ""  
ECHO. & ECHO Done!  
ECHO. & PAUSE
```

### Kill Process

---

#### TASKKILL

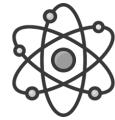
```
:: -----  
:: LAB SETUP  
MD C:\Batch  
Start "" notepad.exe  
CLS  
:: -----  
  
TASKKILL /F /IM Notepad.exe  
ECHO. & ECHO Done!  
ECHO. & PAUSE
```

### Detect Running Process & Wait

---

#### FIND | TASKLIST | TIMEOUT

```
:: -----  
:: LAB SETUP  
MD C:\Batch & Start "" notepad.exe  
CLS  
:: -----
```



```
@ECHO OFF

CLS
ECHO Waiting for application to close...
:: Adds Timing
TIMEOUT /T 3 >nul

:WAIT
TIMEOUT /T 5 >nul
:: With tasklist and find, we can determine if notepad is running.
TASKLIST /FI "IMAGENAME eq notepad.exe" | FIND /I "notepad.exe" >nul
IF %ERRORLEVEL% EQU 0 GOTO :WAIT
GOTO :DONE

:DONE
ECHO Do other stuff here.

ECHO. & ECHO Done!
ECHO. & PAUSE
```

## Detect Running Process & Proceed

### FIND | TASKLIST | TIMEOUT

```
:: Run script and then open Notepad.
@ECHO OFF
CLS
ECHO Detecting if the application is running...
TIMEOUT /T 3 >nul

:WAIT
TIMEOUT /T 5 >nul
:: Using TASKLIST and FIND, we can detect a process.
TASKLIST /FI "IMAGENAME eq notepad.exe" | FIND /I "notepad.exe" >nul
IF %ERRORLEVEL% NEQ 0 GOTO :WAIT
GOTO :DONE

:DONE
ECHO Do other stuff here.

ECHO. & ECHO Done!
ECHO. & PAUSE
```



### One Liner: Return True or False

```
TASKLIST /FI "IMAGENAME eq notepad.exe" | FIND /I "notepad.exe" >nul && (ECHO True) || (ECHO False) & PAUSE
```

### Wait for Multiple Processes to End

**FIND | FOR | TASKLIST | TIMEOUT**

```
@ECHO OFF
CLS

:WAIT
CLS
ECHO Waiting for applications to finish installing...
:: Adds timing to the app installs.
TIMEOUT /T 30 >nul

:: Detect Running Apps
SET "appFound="
FOR %%A IN (app1.exe setup1.exe install1.exe) DO (
    :: Using TASKLIST and FIND, we can detect a process.
    TASKLIST /FI "IMAGENAME eq %%A" | FIND /I "%%A" >nul && SET "appFound=1"
)

:: If any application is still running, loop back to WAIT.
IF DEFINED appFound GOTO :WAIT

:: Proceed when all apps are closed.
GOTO :NEXT

:NEXT
ECHO Do other stuff here.

ECHO .
ECHO Done!

ECHO .
PAUSE
```



## Capture and Use PID to Terminate

### FOR | TASKKILL | TASKLIST | TIMEOUT

A Process ID (PID) is a unique identifier assigned by the operating system to each running process. Every time a program or script is executed, the operating system creates a process to handle that task, and the process is assigned a PID. The PID allows the operating system and other software to manage and track the execution of processes.

```
:: -----
:: LAB SETUP
Taskkill /f /im notepad.exe
Start "" notepad.exe
CLS
:: -----

@ECHO OFF
SETLOCAL
SET procId=
SET "procName=Notepad.exe"

:: Get the Process ID (PID) of notepad.exe.
:: Using TASKLIST in a FOR loop, we can return a process ID.
FOR /F "tokens=2" %%A IN ('TASKLIST /FI "IMAGENAME eq %procName%" /NH') DO SET
"procId=%%A"

:: Check if the process ID is set.
IF NOT DEFINED procId GOTO :END

:RUN
ECHO Process ID of %procName%: %procId%
TIMEOUT /T 10

:: Kill the process by PID.
TASKKILL /F /PID %procId%

:END
ENDLOCAL

ECHO. & ECHO Done!
ECHO. & PAUSE

EXIT /B 0
```



## Batch: Registry Operations

---

How to test from the System Account ([PsExec](#)):

```
PsExec64 -S CMD  
"C:\Batch\script.cmd"
```

### Add Reg Key

---

```
-----  
REG
```

```
@ECHO OFF  
  
:: Create Key  
REG ADD "HKEY_CURRENT_USER\Software\TEST" /F /REG:64  
:: Or /REG:32  
  
:: Create Key with Value and Data  
REG ADD "HKEY_CURRENT_USER\Software\TEST" /V "myKey" /T REG_SZ /D 1 /F /REG:64  
  
ECHO. & PAUSE
```

### Delete Reg Key

---

```
-----  
REG
```

```
:: -----  
:: LAB SETUP  
REG ADD "HKEY_CURRENT_USER\Software\TEST" /V "myKey" /T REG_SZ /D 1 /F /REG:64  
CLS  
:: -----  
  
@ECHO OFF  
  
REG DELETE "HKEY_CURRENT_USER\Software\TEST" /F /REG:64  
:: Or /REG:32  
  
ECHO.  
PAUSE
```



## Delete Reg Key Value

### REG

```
:: -----
:: LAB SETUP
REG ADD "HKEY_CURRENT_USER\Software\TEST" /V "myKey" /T REG_SZ /D 1 /F /REG:64
CLS
:: -----

@ECHO OFF

REG DELETE "HKEY_CURRENT_USER\Software\TEST" /V "myKey" /F /REG:64
:: Or /REG:32

ECHO. & PAUSE
```

## Read Reg Key Value

### ENABLEDELAYEDEXPANSION | FOR | REG

```
:: -----
:: LAB SETUP
REG ADD "HKEY_CURRENT_USER\Software\TEST" /V "myKey" /T REG_SZ /D 1 /F /REG:64
CLS
:: -----

@ECHO OFF

SETLOCAL EnableDelayedExpansion

:: Query the registry key and store the value in a variable.
FOR /F "tokens=3" %%A IN ('REG QUERY "HKEY_CURRENT_USER\Software\TEST" /V
"myKey" /REG:64 2^>nul') DO (
    SET "regValue=%%A"
)

:: Output the value (or handle if not found).
IF DEFINED regValue (
    ECHO Registry Value: !regValue!
) ELSE (
    ECHO Registry key or value not found.
)

ENDLOCAL

ECHO. & PAUSE
```



## Detect Reg Key Value

### REG

```
:: -----
:: LAB SETUP
REG ADD "HKEY_CURRENT_USER\Software\TEST" /V "myKey" /T REG_SZ /D 1 /F /REG:64
CLS
:: -----

@ECHO OFF

:: Detect if a registry key value exists.
REG QUERY "HKEY_CURRENT_USER\Software\TEST" /REG:64 2>nul >nul
IF %ERRORLEVEL% EQU 0 (
    ECHO Registry key value exists.
) ELSE (
    ECHO Registry key value does not exist.
)

ECHO. & PAUSE
```

## Detect Reg Key Value, Return True or False

### FIND | REG

```
:: -----
:: LAB SETUP
REG ADD "HKEY_CURRENT_USER\Software\TEST" /V "myKey" /T REG_SZ /D 1 /F /REG:64
CLS
:: -----

@ECHO OFF

:: The operators '&&' and '||' allow us to simulate a Boolean expression.
REG QUERY "HKEY_CURRENT_USER\Software\TEST" /V "myKey" /REG:64 2>nul >nul &&
(SET "BOOL=TRUE") || (SET "BOOL=FALSE")

ECHO %BOOL%

ECHO.
PAUSE
```



## Detect Reg Key Data, Return True or False

### FIND | REG

```
:: -----
:: LAB SETUP
REG ADD "HKEY_CURRENT_USER\Software\TEST" /V "myKey" /T REG_SZ /D 1 /F /REG:64
CLS
:: -----

@ECHO OFF

:: We're checking the reg key that was created in the lab setup.
REG QUERY "HKEY_CURRENT_USER\Software\TEST" /V "myKey" /REG:64 2>nul | FIND "1"
>nul && (SET "BOOL=TRUE") || (SET "BOOL=FALSE")

ECHO %BOOL%

ECHO. & PAUSE
```

## Scan Reg for String

### FINDSTR | FOR | REG

```
@ECHO OFF

SET "RETURN=FALSE"
SET "STRING=DisableRepair"

:: We're checking a well-known reg key. Change it to detect other keys.
FOR /F "tokens=*" %%A IN ('REG QUERY
"HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Active Setup" /REG:64') DO (
    ECHO %%A | FINDSTR /C:"%STRING%" >nul && (
        SET "RETURN=TRUE"
    )
)

ECHO %RETURN%
ECHO. & PAUSE
```

## NOTE

What's the difference between our *detect* and *scan* examples? The first code example is specific to *one* registry key-value pair, while the second one performs a broader search for a string across *multiple* entries.



## Return Reg Subkeys Based on Data

[FINDSTR](#) | [FOR](#) | [REG](#)

Queries the registry under HKLM\SOFTWARE\Test for all keys and values recursively (/S) in 64-bit view (/REG:64). It processes the results line by line using the FOR /F loop, parsing up to four tokens (%%A, %%B, %%C, and %%D). It checks if the third token (%%C, the value data) contains the string "Acrobat" using FINDSTR. If it does, it outputs the full registry path (HKLM\SOFTWARE\Test\%%A). The script then pauses for user input after running. Think about what you could use this for. Based upon a reg value's data, you could perform an action.

```
:: -----
:: LAB SETUP
REG ADD "HKLM\SOFTWARE\TEST" /V "myKey" /T REG_SZ /D "Acrobat" /F /REG:64
REG ADD "HKLM\SOFTWARE\TEST\TEST2" /V "myKey2" /T REG_SZ /D "Acrobat" /F /REG:64
REG ADD "HKLM\SOFTWARE\TEST\TEST2\TEST3" /V "myKey3" /T REG_SZ /D "Acrobat" /F /REG:64
REG ADD "HKLM\SOFTWARE\TEST\TEST4" /V "myKey4" /T REG_SZ /D "notepad" /F /REG:64
CLS
:: -----

@ECHO OFF

:: Returns the subkeys from our Acrobat condition (from reg data).
ECHO This is quite useful in the automation world.
ECHO.
FOR /F "tokens=1-4" %%A IN ('REG QUERY "HKLM\SOFTWARE\Test" /S /REG:64 2^>nul')
DO (
@ECHO %%C | FINDSTR /C:"Acrobat" >nul && ECHO "HKLM\SOFTWARE\Test\%%A"
)

ECHO. & PAUSE
```

## Examples

**Instead of:**

```
ECHO "HKLM\SOFTWARE\Test\%%A"
```

**Try These (What Ifs):**

```
ECHO REG DELETE "HKLM\SOFTWARE\Test\%%A" /F /REG:64
```

```
ECHO REG ADD "HKLM\SOFTWARE\Test\%%A" /V "NewVal" /T REG_SZ /D "NewData" /F
/REG:64
```



## Add Reg Key for All Users

[ENABLEDELAYEDEXPANSION](#) | [FIND](#) | [FOR](#) | [REG](#)

**Scope:** Admin, System Account, PC, VDI

Add a reg key to each User SID in HKEY\_USERS. Yes this is powerful.

```
@ECHO OFF
SETLOCAL EnableDelayedExpansion

CLS

:: Return User SID
FOR /F "tokens=3" %%A IN ('REG QUERY
"HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Authentication\LogonUI" /V
"SelectedUserSID" /REG:64 2^>nul ^| FIND "REG_SZ"') DO (
    SET "uSID=%%A"
    ECHO Adding key to HKEY_USERS\!uSID!\Software\TEST...
    :: Using the returned user SID, we now add a reg into HKEY_USERS.
    REG ADD "HKEY_USERS\!uSID!\Software\TEST" /V "myKey" /D 1 /F /REG:64 >nul
)

ECHO. & PAUSE
```

## Delete Reg Key for All Users

[ENABLEDELAYEDEXPANSION](#) | [FIND](#) | [FOR](#) | [REG](#)

**Scope:** Admin, System Account, PC, VDI

Delete a reg key from each User SID in HKEY\_USERS. Yes this is powerful.

```
:: -----
:: LAB SETUP
SETLOCAL EnableDelayedExpansion
FOR /F "tokens=3" %%A in ('REG QUERY
"HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Authentication\LogonUI" /V
"SelectedUserSID" /REG:64 2^>nul ^| FIND "REG_SZ"') DO (
    SET "uSID=%%A"
    Reg ADD "HKEY_USERS\!uSID!\Software\TEST" /V "myKey" /D 1 /F /REG:64 >nul
)
CLS
:: -----
```



```
@ECHO OFF  
  
CLS  
  
SETLOCAL EnableDelayedExpansion  
  
:: Return User SID  
FOR /F "tokens=3" %%B IN ('REG QUERY  
"HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Authentication\LogonUI" /V  
"SelectedUserSID" /REG:64 2>nul ^| FIND "REG_SZ"') DO (  
    SET "uSID=%%B"  
    ECHO Deleting key from HKEY_USERS\!uSID!\Software\Test...  
    :: Using the returned user SID, we now delete a reg from HKEY_USERS.  
    REG DELETE "HKEY_USERS\!uSID!\Software\Test" /V "myKey" /F /REG:64 >nul  
)  
  
ENDLOCAL  
  
ECHO. & PAUSE
```

## Detect Reg Key, GOTO Label

[FIND](#) | [GOTO](#) | [REG](#)

### True Condition

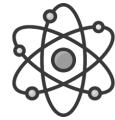
```
REG QUERY "HKEY_CURRENT_USER\Software\Microsoft\Edge" /V "InstallerPinned"  
/REG:64 2>nul | FIND "0" >nul && GOTO :DONE1
```

### False Condition

```
REG QUERY "HKEY_CURRENT_USER\Software\Microsoft\Edge" /V "InstallerPinned"  
/REG:64 2>nul | FIND "0" >nul || GOTO :DONE2
```

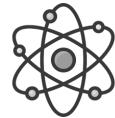
### True or False Condition

```
REG QUERY "HKEY_CURRENT_USER\Software\Microsoft\Edge" /V "InstallerPinned"  
/REG:64 2>nul | FIND "0" >nul && (ECHO TRUE & GOTO :DONE1) || (ECHO FALSE &  
GOTO :DONE2)
```



## Our Script

```
@ECHO OFF  
  
CLS  
  
:: Return True or False, and do something based on that logic.  
REG QUERY "HKEY_CURRENT_USER\Software\Microsoft\Edge" /v "InstallerPinned"  
/REG:64 2>nul | FIND "0" >nul && (ECHO TRUE & GOTO :DONE1) || (ECHO FALSE &  
GOTO :DONE2)  
  
:DONE1  
ECHO Done 1  
ECHO Reg Key exists.  
  
ECHO. & PAUSE  
EXIT  
  
:DONE2  
ECHO Done 2  
ECHO Reg Key does not exist.  
  
ECHO. & PAUSE  
EXIT
```



## Batch: Logical Constructs

---

How to test from the System Account ([PsExec](#)):

```
PsExec64 -S CMD  
"C:\Batch\script.cmd"
```

### IF Condition, Control Flow

---

#### IF

The **IF Condition** is a control flow statement used in programming to execute a block of code only if a specified condition is true. It allows programs to make decisions based on certain conditions.

```
@ECHO OFF  
:: Define the process name.  
SET "procName=notepad.exe"  
  
:: Check if the process is running.  
TASKLIST | FIND /I "%procName%" >nul  
IF NOT ERRORLEVEL 1 (  
    ECHO %procName% is running.  
)  
  
PAUSE
```

### TEMPLATE

---

**Comparison:** `IF "%VARIABLE%" == "value" (Your Code Here)`

**Errorlevel:** `IF ERRORLEVEL ErrorCode (Your Code Here)`

**Existence:** `IF EXIST filename (Your Code Here)`



## IF-ELSE Condition, Control Flow

---

### IF-ELSE

The **IF-ELSE Condition** is a control flow statement in programming that allows the execution of a block of code based on whether a specified condition is true or false. If the condition evaluates to true, one block of code is executed; otherwise, an alternative block of code is executed. This structure enables decision-making in programs by defining different actions depending on the outcome of the condition. Mastering this conditional logic will allow you to create powerful scripts and programs.

```
@ECHO OFF

SET "Flag=FALSE"
IF [%Flag%]==[FALSE] (
    ECHO Flag is set to FALSE.
    ECHO Doing this...
    TIMEOUT /T 3 >nul
) ELSE (
    ECHO Flag is set to TRUE.
    ECHO Doing something else...
    TIMEOUT /T 3 >nul
)

ECHO. & PAUSE
```

---

### TEMPLATE

---

```
IF CONDITION (
    :: TRUE
    Your Code Here
) ELSE (
    :: FALSE
    Your Code Here
)
```



## FOR Loop, Various Implementations

---

### FOR

A **FOR Loop** in batch scripting is a control structure used to iterate over a set of items or perform repetitive tasks. It allows you to execute a block of code for each item in a collection, for every file matching a pattern, or within a specified range.

```
@ECHO OFF

:: The parentheses contain items we want to cycle through.
:: These items could be computer names, servers, users, etc.
FOR %%A IN (User1 User2 User3) DO (
    ECHO Current Username: %%A
    ECHO Your Code Here
    ECHO. & TIMEOUT /T 3 >nul
)

ECHO.
PAUSE
```

---

### TEMPLATE

---

**File Iteration:** FOR %%F IN (\*.txt) DO (Your Code Here)

**Range Iteration:** FOR /L %%N IN (start,step,end) DO (Your Code Here)

**Recursive Traversal:** FOR /R [path] %%F IN (\*.\*) DO (Your Code Here)

**Token Parse:** FOR /F "tokens=1,2 delims=," %%A IN (file.txt) DO (Your Code Here)



## GOTO, Decision-making

---

### GOTO

A **GOTO Statement** allows you to jump to a specific labeled section within a batch script, altering the script's flow. It is often used for breaking out of loops, skipping code blocks, or handling errors.

```
@ECHO OFF

:: Define the USB drive letter and backup folder.
SET "USBDrive=E:"
SET "BackupFolder=C:\Batch"

:: Check if the USB drive exists.
IF NOT EXIST "%USBDrive%" GOTO :NOUSB

:: Perform the backup.
ECHO USB drive detected. Starting backup...
XCOPY "%BackupFolder%" "%USBDrive%\Backup\" /E /I /Y
ECHO Backup completed successfully. & GOTO :DONE

:NOUSB
:: Handle the case where the USB drive is not found.
ECHO Error: USB drive not detected. Please insert the USB drive and try again.

:DONE
ECHO.
ECHO Done!

ECHO.
PAUSE
```

---

### TEMPLATE

---

```
GOTO :LABEL

:LABEL
Your Code Here
```



## CALL, Decision-making

---

### CALL

A **CALL Statement** allows you to execute a specific labeled section within the same batch script (or another batch file) as a subroutine and return to the calling point after it completes. It is often used to modularize scripts, reuse code, and maintain a structured flow for tasks such as condition handling, loops, or repetitive operations.

```
:: -----
:: LAB SETUP
(
    ECHO @ECHO OFF
    ECHO ECHO This is the SubScript called by the main script.
) > "C:\Batch\SubScript.cmd"
CLS
:: -----

@ECHO OFF

ECHO Main script started.

:: Call a secondary script (SubScript.cmd)
CALL C:\Batch\SubScript.cmd

ECHO .
ECHO Done!

ECHO .
PAUSE
```

### TEMPLATE

---

```
CALL :LABEL
:LABEL
Your Code Here
EXIT
```



## FOR Loop with GOTO

### FOR | GOTO

A **FOR Loop** with a **GOTO Statement** allows you to break out of the loop prematurely or redirect the script to another part of the code based on a condition. This can be useful for controlling flow when certain criteria are met during iteration.

```
:: -----
:: LAB SETUP
TASKKILL /F /IM Notepad.exe
START "" Notepad.exe
CLS
:: -----

@ECHO OFF

ECHO Detecting if Notepad is running...
TIMEOUT /T 3 >nul

:: Loop with built-in timer of 10 increments. No counter required.
FOR /L %%A IN (1,1,10) DO (
    CLS
    ECHO %%A
    TASKKILL /F /IM Notepad.exe && GOTO :DOTHIS
    TIMEOUT /T 1 >nul
)

GOTO :DONE

:DOTHIS
ECHO Do stuff here.
GOTO :DONE

:DONE
ECHO Do other stuff here.

ECHO .
ECHO Done!

ECHO .
PAUSE
```



## IF Condition with CALL

-----  
**CALL | IF | TIMEOUT**

An **IF Condition** with a **CALL Statement** is used to evaluate a condition and, based on the result, execute a specific subroutine or external batch file. The **CALL Statement** is particularly useful for running other scripts or invoking labeled sections within the current script.

```
@ECHO OFF

SET "Flag=FALSE"

:: Since our flag is set to false, do this.
IF [%Flag%]==[FALSE] (
    ECHO Let us call a label.
    TIMEOUT /T 3 >nul
    CALL :CALLME
    ECHO You have returned to main processing.
    TIMEOUT /T 3 >nul
)
GOTO :DONE

:CALLME
ECHO Welcome to the CallMe label.
TIMEOUT /T 3 >nul
ECHO Flag is set to FALSE. & TIMEOUT /T 3 >nul
ECHO Do stuff here. & TIMEOUT /T 3 >nul
GOTO :EOF

:DONE
ECHO Do other stuff here.

ECHO. & ECHO Done!
ECHO. & PAUSE
```

### NOTE

---

CALL invokes a subroutine or external script and returns control to the caller after execution. GOTO jumps unconditionally to a label without returning automatically. Use CALL for modularity and reusability, and GOTO for simple redirection.



## Simulate a WHILE LOOP

---

A **WHILE Loop** is a control flow statement used in programming to repeatedly execute a block of code as long as a specified condition is true. It is ideal for scenarios where you don't know in advance how many times the loop will run, but you have a condition to check.

This is our **WHILE Loop** simulation which uses a label and **GOTO Statement**.

```
@ECHO OFF  
  
SET COUNT=1  
:WHILE  
IF %COUNT% LSS 5 (  
    ECHO OUTPUT is %COUNT%  
    SET /A COUNT+=1  
    TIMEOUT /T 1 >nul  
GOTO :WHILE  
)  
  
ECHO.  
ECHO Done!  
  
ECHO.  
PAUSE
```



## Batch: Data Structures

---

### Array Simulation

**ENABLEDELAYEDEXPANSION | FOR | TIMEOUT**

An **Array** is a data structure that stores a *fixed-size* collection of elements, all of the same type, in a contiguous block of memory. Each element in an array can be accessed using its index, which starts from 0 in most programming languages. The idea of 0 actually being 1 or first is the punchline for many programming jokes.

You can simulate **Arrays** using indexed variables.

```
@ECHO OFF

SETLOCAL EnableDelayedExpansion

:: Initial Index Value
SET INDEX=0

:: Array Values
SET "ARRAY=3 1 5 7 9 2 8 9 2 4"

:: Load Array
ECHO Loading our Array...
FOR %%A IN (%ARRAY%) DO (
    ECHO %%A
    SET "ARRAY[!INDEX!]=%%A"
    SET /A INDEX+=1
    CALL :TIMER 1
)
ECHO.

:: Test Array - Return Values
ECHO Retrieving some elements by index...
ECHO Array indEx:0 Element 1: %ARRAY[0]%
ECHO Array indEx:6 Element 7: %ARRAY[6]%
ECHO Array indEx:2 Element 3: %ARRAY[2]%
CALL :TIMER 1

ENDLOCAL
```



```
ECHO.  
ECHO Done!  
ECHO.
```

```
PAUSE  
EXIT
```

```
:TIMER  
:: This is our timer function.  
:: You can use the CALL command to improve  
:: code aesthetics, processing, and output.  
:: You can call other scripts and labels.  
:: Notice how we use seconds.  
  
SET "Seconds=%~1"  
TIMEOUT /T %Seconds% >nul  
:: Good to use in the system account: ping localhost >nul  
GOTO :EOF
```



## Stack Simulation

### ENABLEDELAYEDEXPANSION

A **Stack** follows the Last In, First Out (LIFO) principle. You can simulate a **Stack** by using a combination of set and a counter to manage push and pop operations.

```
@ECHO OFF
SETLOCAL EnableDelayedExpansion

:: Initialize Stack
ECHO Initializing the stack... & CALL :TIMER 1
SET STACK_TOP=0

:: Push Elements to Stack
ECHO Pushing elements to the stack... & CALL :TIMER 1
SET /A STACK_TOP+=1
ECHO FIRST & CALL :TIMER 1
SET STACK[%STACK_TOP%]=FIRST
SET /A STACK_TOP+=1
ECHO SECOND & CALL :TIMER 1
SET STACK[%STACK_TOP%]=SECOND
SET /A STACK_TOP+=1
ECHO THIRD & TIMEOUT /T 1 >nul
SET STACK[%STACK_TOP%]=THIRD
ECHO.

:: Pop Elements from Stack
ECHO Popping elements from stack... & CALL :TIMER 1
ECHO POPPED: !STACK[%STACK_TOP!]! & CALL :TIMER 1
SET /A STACK_TOP-=1
ECHO POPPED: !STACK[%STACK_TOP!]! & CALL :TIMER 1
SET /A STACK_TOP-=1
ECHO POPPED: !STACK[%STACK_TOP!]! & CALL :TIMER 1
SET /A STACK_TOP-=1
ECHO.
ENDLOCAL

ECHO Elements have been processed LIFO! & CALL :TIMER 1

ECHO. & ECHO Done!
ECHO. & PAUSE
EXIT
```



```
:TIMER
:: This is our timer function.
:: You can use the CALL command to improve
:: code aesthetics, processing, and output.
:: You can call other scripts and labels.
:: Notice how we use seconds.

SET "Seconds=%~1"
TIMEOUT /T %Seconds% >nul
:: Good to use in the system account: ping localhost >nul
GOTO :EOF
```



## Queue Simulation

### ENABLEDELAYEDEXPANSION

A **Queue** follows the First In, First Out (FIFO) principle. You can simulate this by managing an index for the "front" and "rear" of the queue.

```
@ECHO OFF
SETLOCAL EnableDelayedExpansion

:: Initialize Queue
ECHO Initializing the queue... & CALL :TIMER 1
SET FRONT=0
SET REAR=0

:: Enqueue Elements
ECHO Enqueuing elements into the queue... & CALL :TIMER 1
SET /A REAR+=1
ECHO FIRST & CALL :TIMER 1
SET QUEUE[%REAR%]=FIRST
SET /A REAR+=1
ECHO SECOND & CALL :TIMER 1
SET QUEUE[%REAR%]=SECOND
SET /A REAR+=1
ECHO THIRD & CALL :TIMER 1
SET QUEUE[%REAR%]=THIRD
ECHO.

:: Dequeue Elements
ECHO Dequeueing elements from the queue... & CALL :TIMER 1
SET /A FRONT+=1
ECHO DEQUEUED: !QUEUE[%FRONT%]! & CALL :TIMER 1
SET /A FRONT+=1
ECHO DEQUEUED: !QUEUE[%FRONT%]! & CALL :TIMER 1
SET /A FRONT+=1
ECHO DEQUEUED: !QUEUE[%FRONT%]! & CALL :TIMER 1
ECHO.

ENDLOCAL

ECHO Elements have been processed FIFO! & CALL :TIMER 1

ECHO. & ECHO Done!
ECHO. & PAUSE
EXIT
```



```
:TIMER
:: This is our timer function.
:: You can use the CALL command to improve
:: code aesthetics, processing, and output.
:: You can call other scripts and labels.
:: Notice how we use seconds.

SET "Seconds=%~1"
TIMEOUT /T %Seconds% >nul
:: Good to use in the system account: ping localhost >nul
GOTO :EOF
```



## Hash Table Simulation

### ENABLEDELAYEDEXPANSION | TIMEOUT

A **Hash Table** is a data structure that stores key-value pairs. It uses a hashing function to compute an index for each key, which determines where the corresponding value is stored in the table. This allows for fast retrieval, insertion, and deletion of data, often in constant time.

You can simulate a **Hash Table** by using variable names with unique keys.

```
@ECHO OFF
SETLOCAL EnableDelayedExpansion

:: Initialize Hash Table
ECHO Initializing the hash table... & CALL :TIMER 1

:: Add Key-Value Pairs
ECHO Adding key-value pairs to the hash table... & CALL :TIMER 1
SET HASH[key1]=Value1
ECHO Key: key1, Value: Value1 & CALL :TIMER 1
SET HASH[key2]=Value2
ECHO Key: key2, Value: Value2 & CALL :TIMER 1
SET HASH[key3]=Value3
ECHO Key: key3, Value: Value3 & CALL :TIMER 1
ECHO.

:: Retrieve Values by Keys
ECHO Retrieving values from the hash table... & CALL :TIMER 1
ECHO Key: key1, Retrieved Value: !HASH[key1]! & CALL :TIMER 1
ECHO Key: key2, Retrieved Value: !HASH[key2]! & CALL :TIMER 1
ECHO Key: key3, Retrieved Value: !HASH[key3]! & CALL :TIMER 1
ECHO.

:: Update Value in Hash Table
ECHO Updating a value in the hash table... & CALL :TIMER 1
SET HASH[key2]=UpdatedValue2
ECHO Key: key2, New Value: !HASH[key2]! & CALL :TIMER 1
ECHO.

:: Remove Key-Value Pair
ECHO Removing a key-value pair from the hash table... & CALL :TIMER 1
SET HASH[key3]=
ECHO Key: key3 has been removed. Retrieved Value: !HASH[key3]! (Empty means removed) & CALL :TIMER 1
```



```
ECHO.  
ENDLOCAL  
  
ECHO Hash table operations completed! & CALL :TIMER 1  
  
ECHO. & ECHO Done!  
ECHO. & PAUSE  
EXIT  
  
:TIMER  
:: This is our timer function.  
:: You can use the CALL command to improve  
:: code aesthetics, processing, and output.  
:: You can call other scripts and labels.  
:: Notice how we use seconds.  
  
SET "Seconds=%~1"  
TIMEOUT /T %Seconds% >nul  
:: Good to use in the system account: ping localhost >nul  
GOTO :EOF
```



## Batch: Miscellaneous

---

How to test from the System Account ([PsExec](#)):

```
PsExec64 -S CMD  
"C:\Batch\script.cmd"
```

### Create a Counter, 1-10

---

```
@ECHO OFF  
  
SET counter=1  
  
:LOOP  
ECHO %counter%  
TIMEOUT /T 1 >nul  
SET /A counter+=1  
IF %counter% GTR 10 PAUSE & EXIT  
GOTO :LOOP
```

### Create a Countdown Timer, 10-1

---

```
@ECHO OFF  
  
SET COUNTER=10  
  
:LOOP  
ECHO %COUNTER%  
TIMEOUT /T 1 >nul  
SET /A COUNTER-=1  
IF %COUNTER% LSS 1 PAUSE & EXIT  
GOTO :LOOP
```

### Detect FSLogix

---

```
REG  
  
@ECHO OFF  
  
SET "fslogixPath=C:\Program Files\FSLlogix"
```



```
:: Detect FSLogix
IF EXIST "%fslogixPath%" (
    SET "FsLogix=TRUE"
) ELSE (
    SET "FsLogix=FALSE"
)
ECHO %FsLogix%
ECHO. & PAUSE
```

**Or**

```
@ECHO OFF

SET "regKey=HKLM\SOFTWARE\FSLogix"
SET "FsLogix=FALSE"
:: Check if the registry key exists.
REG QUERY "%regKey%" >nul 2>&1
IF %ERRORLEVEL% EQU 0 (
    SET "FsLogix=TRUE"
)

ECHO %FsLogix%
ECHO. & PAUSE
```

## Hide Output of Command

---

**TASKKILL**

```
@ECHO OFF

ECHO Normal Output:
START "" Notepad.exe & TIMEOUT /T 2 >nul
TASKKILL /F /IM Notepad.exe
ECHO.

START "" Notepad.exe
ECHO Hidden Output: & ECHO. & TIMEOUT /T 2 >nul
TASKKILL /F /IM Notepad.exe >nul 2>&1

ECHO.
PAUSE
```



## Set PowerShell Remote Signed

### SET-EXECUTIONPOLICY

```
@ECHO OFF  
  
:: Executes as 64 Bit  
  
:: Using an admin path will force a 32 bit compiled script to run the  
:: command as 64 bit.  
\\%COMPUTERNAME%\C$\Windows\system32\WindowsPowerShell\v1.0\PowerShell.exe  
-Command "Set-ExecutionPolicy RemoteSigned"  
  
ECHO. & PAUSE
```

## Launch PowerShell Script.ps1

```
@ECHO OFF  
  
:: Execute as 64 bit  
:: Using an admin path will force a 32 bit compiled script to run the  
:: command as 64 bit.  
START "" /B  
\\%COMPUTERNAME%\C$\Windows\system32\WindowsPowerShell\v1.0\PowerShell.exe  
-File .\script.ps1  
  
\\%COMPUTERNAME%\C$\Windows\system32\WindowsPowerShell\v1.0\PowerShell.exe  
-NoProfile -WindowStyle Hidden -ExecutionPolicy Bypass -File .\script.ps1  
  
ECHO. & PAUSE
```

## Return OS Major Version

### VER

**VER.exe** is a command-line utility that displays the version number of the operating system.

```
@ECHO OFF  
  
FOR /F "tokens=1,2,3,4 delims=[.]" %%A IN ('VER') DO SET "wBuild=%%D"  
  
ECHO Build: %wBuild%  
  
ECHO. & PAUSE
```



## Detect if Current Computer is Online

**COLOR | FIND | PING**

```
:: Compact
PING -n 2 www.google.com | FIND /I "bytes=" >nul && (SET BOOL=TRUE) || (SET
BOOL=FALSE)
ECHO %BOOL% & PAUSE
```

**Or**

```
@ECHO OFF

CD "%~dp0"

TITLE Detect Current Computer Online Status
CLS
COLOR 0A
```

```
ECHO.
ECHO Detecting if the current computer is online...
ECHO. & PING -n 2 -w 1000 localhost >nul
```

```
:: Perform a self-check using ping.
PING -n 2 -4 www.google.com | FIND /I "bytes=" >nul
IF ERRORLEVEL 1 (
    ECHO The current computer is OFFLINE.
) ELSE (
    ECHO The current computer is ONLINE.
)

ECHO.
PAUSE
```



## Detect if Computers are Online

### FIND | FOR

Checks the connectivity of a predefined list of computers (pcList) by pinging each one and identifying which are online. It iterates through the list, pings each computer, and adds the names of the responsive ones to an onlinePCs list. After completing the scan, it displays the names of all online computers and concludes with a "Done!" message before pausing for the user's review.

```
@ECHO OFF

:: Define Variables
:: Add - Remove PCs Here. You could also load computers from computers.txt.
SET "pcList=PC1 PC2 PC3 PC4"

CD "%~dp0"

TITLE Detect Online Computers
CLS
COLOR 0A
SETLOCAL EnableDelayedExpansion
SET pcName=
SET onlinePCs=

ECHO.
ECHO Detecting online devices...
ECHO. & PING -n 2 -w 1000 localhost >nul

FOR %%V IN (%pcList%) DO (
    ECHO Testing %%V...
    FOR /F "tokens=*%R IN ('PING -n 1 %%V -4 ^| FIND /I "bytes"') DO (
        SET "onlinePCs=%V !onlinePCs!"
    )
)

ECHO Online PCs: %onlinePCs%
ECHO. & PING -n 2 -w 1000 localhost >nul
ECHO.
SET "onlinePCs=!onlinePCs:~1!"
ENDLOCAL

ECHO. & ECHO Done!
ECHO. & PAUSE
```



## VDI Engine, Cycle VM List

**COLOR | ENABLEDELAYEDEXPANSION | FIND | FOR | FINDSTR | REG | SC**

Detects online virtual machines (VMs) from a predefined list, retrieves user SIDs from each online VM, and logs their details to a report file. It begins by clearing any existing report file and scanning the VM list to identify which are online. For each online VM, it queries the registry remotely to find user SIDs and associated usernames, then performs remote actions while logging the VM name, username, and SID to the report. The script concludes by displaying a "Done!" message and pausing for user review.

```
@ECHO OFF

:: Define Variables
:: Add - Remove VMs Here
SET "vmList=VM1 VM2 VM3 VM4"
SET "reportPath=Report.txt"

CD "%~dp0"

TITLE Azure VDI Engine
CLS
COLOR 0A
SETLOCAL EnableDelayedExpansion

SET uName=
SET vmName=
SET uSID=
SET onlineVMs=
SET "sidPattern1=S-1-5-21-"

IF EXIST %reportPath% DEL /Q %reportPath%

ECHO AZURE VDI ENGINE
ECHO.
ECHO Detecting online VMs...
ECHO. & PING -n 6 -w 1000 localhost >nul

FOR %%V IN (%vmList%) DO (
    FOR /F "tokens=*%R IN ('PING -n 1 %%V ^| FIND /I "bytes"') DO (
        SET "onlineVMs=%V !onlineVMs!"
        :: SC \\%%V config RemoteRegistry start= auto >nul 2>&1
        :: SC \\%%V start RemoteRegistry >nul 2>&1
    )
)
```



```
ECHO Online VMs: %onlineVMs%
ECHO. & PING -n 6 -w 1000 localhost >nul
SET "onlineVMs=!onlineVMs:~1!"
ECHO.

:: DYNAMICALLY BUILD USER SIDs AND RUN ACTIONS
ECHO Running remote actions...
ECHO. & PING -n 6 -w 1000 localhost >nul

:: RETURN USER SIDS

:: Using online VMs, return each user's SID, remotely.
FOR %%H IN (%onlineVMs%) DO (
    SET "vmName=%%H"
    FOR /F "tokens=1,2,* delims=" %%S IN ('REG QUERY "\\\!vmName!\HKEY_USERS"
/REG:64 2^>nul ^| FINDSTR /R /C:"!sidPattern1!"') DO (
        ECHO %%S | FINDSTR /R /C:_Classes" >nul
        IF ERRORLEVEL 1 (
            SET "uSID=%%S"
            FOR /F "tokens=1,2,3" %%D IN ('REG QUERY
"\\\!vmName!\!uSID!\Volatile Environment" /V "USERNAME" /REG:64 2^>nul') DO (SET
"uName=%%F")
            CALL :REMOTE
        )
    )
    SET vmName=
    SET uName=
    SET uSID=
    ECHO.
)
ENDLOCAL

ECHO. & ECHO Done!
ECHO. & PAUSE
EXIT

:REMOTE
ECHO !vmName! !uName! !uSID!
ECHO !vmName! !uName! !uSID! >> %reportPath%
PING -n 2 -w 1000 localhost >nul
GOTO :EOF
```



## Clear All Windows Logs

### FOR | WEVTUTIL

**WEVTUTIL.exe** (Windows Event Utility) is a command-line tool in Windows used to manage event logs and publishers. It provides comprehensive control over the event logs, allowing administrators to query, export, archive, and configure logs programmatically or through scripting.

```
:: Requires Admin Privileges  
  
@ECHO OFF  
  
FOR /F "tokens=*" %%1 IN ('WEVTUTIL el') DO WEVTUTIL cl "%~1"  
  
ECHO. & PAUSE  
:: export log: wevtutil epl Application C:\Batch\Application.evtx
```

## Import Certificate

### CERTUTIL

**CERTUTIL.exe** is a command-line utility included with Windows that provides administrators with a range of tools for managing digital certificates and the Windows Certificate Store. It's part of the Windows Certificate Services and is commonly used for tasks related to Public Key Infrastructure (PKI).

```
@ECHO OFF  
  
:: Define Variables  
SET "certCer=Your-Certificate.cer"  
SET "certPfx=Your-Certificate.pfx"  
SET "certPw=Cert-Password-Here"  
  
CERTUTIL -AddStore -Enterprise -F -V CA "%certCer%"  
CERTUTIL -AddStore -Enterprise -F -V Root "%certCer%"  
CERTUTIL -AddStore -Enterprise -F -V TrustedPublisher "%certCer%"  
CERTUTIL -F -USER -P %certPw% -Enterprise -ImportPFX TrustedPeople "%certPfx%"  
CERTUTIL -F -USER -P %certPw% -Enterprise -ImportPFX Root "%certPfx%"  
  
ECHO. & PAUSE
```



## Sign & Verify Executable with Certificate

### SIGNTOOL

**SIGNTOOL.exe** is a command-line tool provided by Microsoft as part of the Windows Software Development Kit (SDK). It is used for signing and verifying digital signatures on files, such as executables, DLLs, and other types of code. The primary purpose of SIGNTOOL.exe is to help developers ensure that their software is trusted by users, particularly when distributed over the internet or used in environments that require signed code for security purposes. Something to note, this tool can be used to sign numerous file types, like PowerShell, APPX, MSIX, and CAB.

```
@ECHO OFF  
  
SIGNTOOL SIGN /F "Your-Certificate.pfx" /P Cert-Password-Here /FD SHA256  
App.exe  
  
SIGNTOOL VERIFY /PA "App.exe"  
  
ECHO. & PAUSE
```

## Add Application to Windows Path

### REG | SETX

```
@ECHO OFF  
  
SET "appName=MyApp.exe"  
SET "appPath=C:\Batch"  
  
REG ADD "HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\App  
Paths\%appName%" /V "Path" /T REG_SZ /D "%appPath%" /F /REG:64  
  
REG ADD "HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\App  
Paths\%appName%" /VE /D "%appPath%\%appName%" /F /REG:64  
  
SET "PATH=%appPath%;%PATH%"  
ECHO %PATH%  
  
:: Persist the PATH change.  
SETX PATH "%appPath%;%PATH%"  
  
ECHO. & PAUSE
```



## Create Network Share, Shared Folder

[FIND](#) | [FOR](#) | [NET](#) | [REG](#)

Retrieves the current user's SID and profile path from the registry, creates a Shared-Folder directory in the user's profile if it doesn't already exist, deletes any existing network share named *Your-Share*, and creates a new share pointing to the *Shared-Folder* with full access for everyone. It ends with a pause so you can see the output.

```
@ECHO OFF

:: Return User SID
FOR /F "tokens=3" %%A IN ('REG QUERY
"HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Authentication\LogonUI" /V
"SelectedUserSID" /REG:64 2>nul ^| FIND "REG_SZ"') DO (SET "uSID=%%A")

:: Return User Profile Path
FOR /F "tokens=2*" %%A IN ('REG QUERY
"HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows
NT\CurrentVersion\ProfileList\%uSID%" /V "ProfileImagePath" /REG:64') DO (SET
"userProfilePath=%%B")

ECHO %userProfilePath%

IF NOT EXIST "%userProfilePath%\Shared-Folder" MD
"%userProfilePath%\Shared-Folder"

:: net share f: /delete, net share ShareName /delete
NET SHARE Your-Share /DELETE

:: Creates a share in the user profile.
NET SHARE Your-Share="%userProfilePath%\Shared-Folder" /GRANT:Everyone,FULL
/REMARK:"Used for sharing content"

ECHO. & PAUSE
```



## Animation (Windows Terminal)

### COLOR | WINDOWS TERMINAL

This is an animation I made for Windows Terminal. Maybe you will find it interesting. Just save as script.cmd, and run in a Windows Terminal.

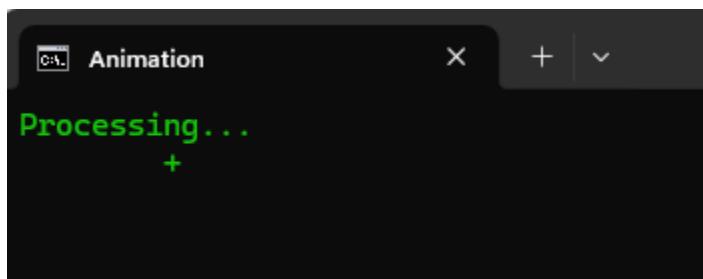
```
@ECHO OFF

TITLE Animation
COLOR 0A

PowerShell -ExecutionPolicy Bypass -NoProfile -Command ^
"$length = 10;" ^
"$pos = 0;" ^
"$direction = 1;" ^
"$line = ' ' * $length;" ^
"$persistentText = 'Processing...';" ^
"DO {" ^
    "CLS;" ^
    "$chars = $line.ToCharArray();" ^
    "$chars[$pos] = '+';" ^
    "$currentLine = -Join $chars;" ^
    "Write-Output $persistentText;" ^
    "Write-Output $currentLine;" ^
    "Start-Sleep -Milliseconds 300;" ^
    "$pos += $direction;" ^
    "if ($pos -eq 0 -or $pos -eq ($length - 1)) { $direction = -$direction; }" ^
"} while (-not $Host.UI.RawUI.KeyAvailable)"

ECHO. & PAUSE
```

### | output |





## Batch: Task Scheduler

---

How to test from the System Account ([PsExec](#)):

```
PsExec64 -S CMD  
"C:\Batch\script.cmd"
```

## On Demand App Install

---

[ICACLS | SCHTASKS](#)

**Scope:** Admin, System Account, PC, VDI

Install enterprise apps using a scheduled task. Create a silent install package, or pass the silent install options to a local setup file. Set the permissions on the task using ICACLS. Once this script has been deployed, and the task has been created, users can install apps as System.

```
@ECHO OFF  
  
:: Define the path to the MSI file.  
SET "MSI_FILE=C:\Batch\setup.msi"  
  
:: Define the name of the scheduled task.  
SET "TASK_NAME=App-Install"  
  
:: Create the scheduled task to run the install command as SYSTEM.  
SCHTASKS /CREATE /TN "%TASK_NAME%" /TR "\"%SystemRoot%\System32\msiexec.exe\"  
/QN /I %MSI_FILE% /SC ONCE /ST 00:00 /RL HIGHEST /RU SYSTEM /F  
  
ICACLS "C:\Windows\System32\Tasks\%TASK_NAME%" /Grant Everyone:(RX)  
  
ECHO Scheduled task created to install the app as SYSTEM.  
ECHO. & PAUSE
```

### NOTE

---

Think about how you could use Task Scheduler to assist with app installs, configuration, repairs, upgrades, and even uninstalls.



## On Demand App Repair

### ICACLS | SCHEDTASKS

Repair enterprise apps using a scheduled task. Create a silent repair package, or pass the silent repair options to a local setup file. Set the permissions on the task using ICACLS. Once this script has been deployed, and the task has been created, users can repair apps as System.

```
@ECHO OFF

:: Define the path to the MSI file.
SET "MSI_FILE=C:\Batch\setup.msi"

:: Define the name of the scheduled task.
SET "TASK_NAME=App-Repair"

:: Create the scheduled task to run the repair command as SYSTEM.
SCHEDTASKS /CREATE /TN "%TASK_NAME%" /TR "\"%SystemRoot%\System32\msiexec.exe\" /QN /FAUMS %MSI_FILE%" /SC ONCE /ST 00:00 /RL HIGHEST /RU SYSTEM /F

ICACLS "C:\Windows\System32\Tasks\%TASK_NAME%" /Grant Everyone:(RX)

ECHO Scheduled task created to repair the app as SYSTEM.
ECHO. & PAUSE
```

## On Demand App Download

### ICACLS | SCHEDTASKS

**Scope:** Admin, System Account, PC, VDI

Download enterprise apps using a scheduled task. Set the permissions on the task using ICACLS. Once this script has been deployed, and the task has been created, users can download apps as System.

```
@ECHO OFF

:: Define the path to an online MSI file.
SET
"DOWNLOAD_FILE=https://dl.google.com/tag/s/dl/chrome/install/googlechromestandaloneenterprise64.msi"

:: Define the name of the scheduled task.
SET "TASK_NAME=App-Download"
```



```
:: Create the scheduled task to run the download command as SYSTEM.  
SCHTASKS /CREATE /TN "%TASK_NAME%" /TR "\"curl\" %DOWNLOAD_FILE% --output  
C:\Batch\setup.msi" /SC ONCE /ST 00:00 /RL HIGHEST /RU SYSTEM /F  
  
ICAcls "C:\Windows\System32\Tasks\%TASK_NAME%" /Grant Everyone:(RX)  
  
ECHO Scheduled task created to repair the app as SYSTEM.  
ECHO. & PAUSE
```

## Restart Print Spooler Every 24 Hours

---

### SCHTASKS

**Scope:** Admin, System Account, PC, VDI

```
@ECHO OFF  
  
SCHTASKS /CREATE /TN "Restart-Print-Spooler" /TR "NET STOP Spooler && NET START  
Spooler" /SC DAILY /ST 02:00 /RL HIGHEST /RU SYSTEM /F  
  
ECHO. & PAUSE
```

## Schedule PowerShell Script to Run

---

### SCHTASKS

**Scope:** Admin, System Account, PC, VDI

```
@ECHO OFF  
  
:: Define the path to the PowerShell script.  
SET "PS_SCRIPT=C:\PowerShell\script.ps1"  
  
:: Define the name of the scheduled task.  
SET "TASK_NAME=Run-PowerShell-Script"  
  
:: Create the scheduled task to run the PowerShell script as SYSTEM.  
SCHTASKS /CREATE /TN "%TASK_NAME%" /TR  
"\\"%SystemRoot%\System32\WindowsPowerShell\v1.0\powershell.exe\" -NoProfile  
-ExecutionPolicy Bypass -File \"%PS_SCRIPT%\" /SC DAILY /ST 06:00 /RL HIGHEST  
/RU SYSTEM /F  
  
ECHO Scheduled task created to run the PowerShell script as SYSTEM.  
  
ECHO. & PAUSE
```



## Schedule PowerShell Command to Run

### SCHTASKS

**Scope:** Admin, System Account, PC, VDI

```
@ECHO OFF

:: Define the PowerShell command.
SET "PS_CMD=if (Test-Path 'C:\PowerShell\File.txt') { Remove-Item
'C:\PowerShell\File.txt' }"

:: Define the name of the scheduled task.
SET "TASK_NAME=Run-PowerShell-Cmd"

:: Create the scheduled task to run the PowerShell command as SYSTEM.
SCHTASKS /CREATE /TN "%TASK_NAME%" /TR
"%SystemRoot%\System32\WindowsPowerShell\v1.0\powershell.exe" -NoProfile
-ExecutionPolicy Bypass -Command "%PS_CMD%" /SC DAILY /ST 06:00 /RL HIGHEST
/RU SYSTEM /F

ECHO.
ECHO Scheduled task created to run the PowerShell command as SYSTEM.

ECHO. & PAUSE
```

## Back up Documents Folder Weekly

### FOR | REG | SCHTASKS

**Scope:** Admin, System Account, PC, VDI

```
@ECHO OFF
SETLOCAL

:: Set the path for the backup script.
SET "scriptPath=C:\Batch\Backup.cmd"

:: Retrieve User SID
FOR /F "tokens=3" %%A IN ('REG QUERY
"HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Authentication\LogonUI" /V
"SelectedUserSID" /REG:64 2>nul ^| FIND "REG_SZ"') DO (SET "uSID=%%A")
```



```
:: Verify SID Retrieval
IF NOT DEFINED uSID (
    ECHO User SID not found. Exiting.
    GOTO :FAIL
)

:: Retrieve User Profile Path
FOR /F "tokens=2*" %%B IN ('REG QUERY
    "HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows
    NT\CurrentVersion\ProfileList\%uSID%" /V "ProfileImagePath" /REG:64') DO (SET
    "uProfile=%%C")

:: Verify Profile Path
IF NOT DEFINED uProfile (
    ECHO User profile not found. Exiting.
    GOTO :FAIL
)

:: Set the Documents directory path.
SET "SOURCE=%uProfile%\Documents"
SET "DEST=C:\Backups\Documents"

:: Create the backup destination directory if it does not exist.
IF NOT EXIST "%DEST%" (
    MD "%DEST%"
)

:: Generate Backup Script
IF NOT EXIST "%scriptPath%" (
    MD "C:\Batch"
    >"%scriptPath%" ECHO @ECHO OFF
    >>%scriptPath% ECHO SET SOURCE=%uProfile%\Documents
    >>%scriptPath% ECHO SET DEST=C:\Backups\Documents
    >>%scriptPath% ECHO IF NOT EXIST "%DEST%" MD "%DEST%"
    >>%scriptPath% ECHO XCOPY "%SOURCE%" "%DEST%" /E /H /C /I /Y
    >>%scriptPath% ECHO ECHO Backup completed at %DATE% %TIME%
    ^>^>%DEST%\backup_log.txt"
)

:: Create a scheduled task to run the generated backup script weekly.
SCHTASKS /CREATE /TN "Backup-Documents" /TR "C:\Batch\Backup.cmd" /SC WEEKLY /D
SUN /ST 02:00 /RL HIGHEST /RU SYSTEM /F

:END
ENDLOCAL
ECHO Success! & PAUSE
EXIT /B 0
```



```
:FAIL  
ENDLOCAL  
ECHO Fail! & PAUSE  
EXIT /B 1
```

## Manually Test URL & Download

Copy/Paste these URLs into your browser or command line. Verify that they work.

### Browser

<https://dl.google.com/tag/s/dl/chrome/install/googlechromestandaloneenterprise64.msi>

### Command Line

```
CURL "https://dl.google.com/tag/s/dl/chrome/install/googlechromestandaloneenterprise64.msi"  
--output "C:\Batch\Chrome.msi"
```

### NOTE

---

There are many direct download URLs that never change, yet the content is updated as the vendor releases newer versions. Think about how that could be used in your scripts and with automation.



## Disk Cleanup 1

### SCHTASKS

**Scope:** Admin, System Account, PC, VDI

```
:: Create a task that performs disk cleanup.  
:: Requires an admin to create the task, but a restricted user  
:: can run it afterwards.  
  
@ECHO OFF  
  
SCHTASKS /CREATE /TN "Weekly-Disk-Cleanup" /TR "CMD /C RD /Q /S %TEMP% &  
cleanmgr /sagerun:1" /SC WEEKLY /D SUN /ST 02:00 /RL HIGHEST /RU SYSTEM /F  
  
ECHO. & PAUSE
```

## Disk Cleanup 2

### SCHTASKS

**Scope:** Admin, System Account, PC, VDI

```
:: Create a task that performs disk cleanup.  
:: Compile as 64 Bit. Can be deployed from SCCM or Intune.  
  
@ECHO OFF  
  
:: Define Variables  
SET "taskName=Weekly-Disk-Cleanup"  
SET "scriptPath=C:\Batch\Temp-Cleanup.cmd"  
  
:: Retrieve User SID  
FOR /F "tokens=3" %%A IN ('REG QUERY  
"HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Authentication\LogonUI" /V  
"SelectedUserSID" /REG:64 2>nul ^| FIND "REG_SZ"') DO (SET "uSID=%%A")  
  
:: Verify  
IF NOT DEFINED uSID (EXIT /B 1)  
:: ECHO %uSID% & PAUSE  
  
:: Retrieve User Profile  
FOR /F "tokens=2*" %%B IN ('REG QUERY  
"HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows  
NT\CurrentVersion\ProfileList\%uSID%" /V "ProfileImagePath" /REG:64') DO (SET  
"uProfile=%C")
```



```
:: Verify
IF NOT DEFINED uProfile (EXIT /B 1)
:: ECHO %uProfile% & PAUSE

:: Set the folder to clean.
SET "uTEMP=%uProfile%\AppData\Local\Temp"

:: Verify
ECHO %uTemp% | FIND "C:\Users" >nul
IF %ERRORLEVEL% EQU 1 EXIT /B 1
:: ECHO %uTemp% & PAUSE
:: ECHO %ERRORLEVEL% & PAUSE

:: Create our cleanup script.
:: Could use this: >> "%scriptPath%" ECHO RD /Q /S "%uTEMP%"
IF NOT EXIST "%scriptPath%" (
    MD "C:\Batch"
    >"%scriptPath%" ECHO @ECHO OFF
    >>"%scriptPath%" ECHO ECHO %uTemp% ^| FIND "AppData\Local\Temp" 2^>nul
    >>"%scriptPath%" ECHO IF ERRORLEVEL 1 EXIT /B 1
    >>"%scriptPath%" ECHO DEL /Q "%uTEMP%\*"
    >>"%scriptPath%" ECHO cleanmgr /sagerun:1)

:: Create a scheduled task to run the cleanup in the system account.
SCHTASKS /CREATE /TN "%taskName%" /TR "%scriptPath%" /SC WEEKLY /D SUN /ST
02:00 /RL HIGHEST /RU SYSTEM /F

ECHO.
ECHO Done!

ECHO.
PAUSE
```



## Monitor High CPU Usage

---

### SCHTASKS

**Scope:** Admin, System Account, PC, VDI

```
:: Monitor and report on high CPU usage.  
:: Restart the service.
```

```
@ECHO OFF
```

```
SCHTASKS /CREATE /TN "CPU-Monitor" /TR "PowerShell.exe -NoProfile -Command \"if  
( (Get-Counter '\\Processor(_Total)\\% Processor  
Time').CounterSamples.CookedValue -gt 90) { Restart-Service -Name 'WinDefend'  
-Force }\" /SC MINUTE /MO 5 /RL HIGHEST /RU SYSTEM /F
```

```
ECHO .
```

```
ECHO Done!
```

```
ECHO .
```

```
PAUSE
```

---

### NOTE

If you want restricted users to be able to run elevated tasks, you need to set the permissions on the task.

```
ICAcls "C:\Windows\System32\Tasks\%TASK_NAME%" /Grant Everyone:(RX)
```



## 6. PowerShell

---

### Brief Overview

**PowerShell** was Microsoft's innovative answer to the longstanding limitations of batch scripting. Developed in 2006 under the leadership of Jeffrey Snover and built on the robust .NET framework, it reimagined what scripting could be—powerful, modern, and surprisingly elegant. With the PowerShell platform, repetitive and complex administrative tasks were transformed from tedious chores into efficient, even enjoyable processes. Well, certainly enjoyable to many of us.

So, what sets PowerShell apart from other scripting environments? Enter CMDlets (pronounced *command-lets*). These lightweight, task-focused commands are like a system administrator's fantasy toolkit come to life. Whether you need to query a system component, configure multiple servers, or automate tasks across diverse platforms, CMDlets are up to the challenge. Their seamless integration with Windows systems and their object-oriented design allow PowerShell to tackle tasks with a level of sophistication and precision that batch files could only dream of achieving.

But CMDlets are just the beginning. PowerShell introduced game-changing innovations such as pipelining, which allows the output of one command to serve as the input for another, fostering an elegant flow of data and processes. It also boasts robust error handling, giving users granular control over how issues are managed. Additionally, its ability to dynamically manipulate objects opens up a world of possibilities for automating and streamlining tasks. Suddenly, scripting became less about struggling with constraints and more about harnessing new possibilities.

Beyond its technical capabilities, PowerShell is celebrated for its versatility. While it's most commonly associated with IT administration—managing servers, automating workflows, and maintaining system security—it has found use in a wide variety of fields. The scripting language's extensibility and cross-platform capabilities (thanks to the open-source PowerShell Core) make it a favorite tool not just for Windows, but also for macOS and Linux users.



Today, PowerShell is indispensable in the IT world and beyond. Whether you're orchestrating the setup of hundreds of virtual machines, writing scripts to automate your organization's workflows, or just managing files on your personal computer, PowerShell empowers you to do more with less effort. It's the versatile, scalable tool that makes everything from routine system administration to ambitious automation projects not only possible, but enjoyable. Who knows? With PowerShell's growing capabilities, automating your morning coffee might be closer than you think.

## Why is it practical today?

- **Cross-Platform Support:** With the introduction of PowerShell Core, it works across Windows, macOS, and Linux.
- **Object-oriented:** Unlike batch files, PowerShell operates on objects rather than text, enabling easier data manipulation.
- **Advanced Features:** Includes error handling, debugging tools, and robust control structures for creating complex scripts.
- **Extensibility:** PowerShell supports modules and integrates with other tools, like Active Directory and Azure, making it highly versatile.
- **Community and Ecosystem:** A large repository of reusable scripts and modules exists in the PowerShell Gallery, helping users solve problems quickly.

## PowerShell vs. Batch Scripting

PowerShell supersedes Batch scripting in functionality, scalability, and complexity. While Batch scripting is limited to sequential commands, PowerShell scripts can interact with APIs, manage cloud environments, and query databases. However, Batch files do remain a go-to for simpler tasks, thanks to their lightweight nature.



## Limitations of PowerShell

- Requires a steeper learning curve for beginners.
- May need installation on older Windows versions or non-Windows systems.
- Scripts may run slower compared to native executables or compiled scripts.
- CMDlets and features can become deprecated, leaving your scripts in a non-working State. {Shakes fist at Microsoft}

Regardless of any limitations, PowerShell's blend of simplicity and power makes it indispensable for modern automation, empowering IT professionals to manage increasingly complex environments with efficiency. If you're only going to learn one scripting language, this is the one you should learn.

When naming your PowerShell scripts, use .ps1 for the extension.

**Examples:** Script.ps1, My\_Script.ps1, Copy-Script.ps1

**Thank you, Mr. Snover, for giving us something truly remarkable.**



## PowerShell: Design

---

<https://learn.microsoft.com/en-us/powershell/scripting/discover-powershell>

### Basic Construction

---

Here we are; here we go. Let's build our first **PowerShell** script. Using your favorite code editor (Notepad++ is a good one to start with), *type* and then *save* this code as *script.ps1* and *run* it. Read the code comments for details, which is usually where I live. If you have any problems running this script, check the execution policy on your computer.

```
# Set Window Title
$host.UI.RawUI.WindowTitle = "Greeting Script"

# Set text and background color (black background, green text).
$host.UI.RawUI.BackgroundColor = "Black"
$host.UI.RawUI.ForegroundColor = "Blue"

# Clear Console Window
Clear-Host

# Display Welcome Message with blue color.
Write-Host "Welcome to PowerShell Scripting!`n"

# Set color to cyan for the input prompt.
$host.UI.RawUI.ForegroundColor = "Cyan"

# Capture user input.
$Name = Read-Host "What is your name?"

# Set the color back to green for the greeting message.
$host.UI.RawUI.ForegroundColor = "Green"

# Display the greeting.
Write-Host "Hello, $Name! Have a great day!"

# Set color to yellow for the pause.
$host.UI.RawUI.ForegroundColor = "Yellow"
```



```
# Pause the script to allow the user to see the message.  
Read-Host "`nPress Enter to continue..." # or pause  
  
# Set color to gray.  
$host.UI.RawUI.ForegroundColor = "Gray"  
  
# Exit our script.  
exit # Exit codes: 1 Normal. 2 Error.
```

## NOTE

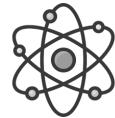
---

### Set Execution Policy

```
Set-ExecutionPolicy RemoteSigned
```

### Reference

<https://learn.microsoft.com/en-us/powershell/module/microsoft.powershell.security/set-executionpolicy>



## PowerShell: String Operations

---

### Remove Leading Space

---

```
# Remove Leading Space
$Var = "    String"
Write-Host "Before: '$Var'"
$Var = $Var.TrimStart()
Write-Host "After: '$Var"'; pause # or Read-Host
```

### Remove Trailing Space

---

```
# Remove Trailing Space
$Var = "String    "
Write-Host "Before: '$Var'"
$Var = $Var.TrimEnd()
Write-Host "After: '$Var"'; pause # or Read-Host
```

### Remove Leading & Trailing Space

---

```
# Remove Leading and Trailing Space
$Var = "    String    "
Write-Host "Before: '$Var'"
$Var = $Var.Trim()
Write-Host "After: '$Var"'; pause # or Read-Host
```

### Detect & Remove Leading Space

---

```
Clear-Host; $host.UI.RawUI.ForegroundColor = "Yellow"

# Detect and Remove Leading Space
$Var = "    String"
if ($Var.StartsWith(" ")) {
```



```
Write-Host "Space Detected: '$Var'"  
$Var = $Var.TrimStart()  
Write-Host "Leading space removed: '$Var'"  
} else {  
    Write-Host "No leading space."  
}  
  
Write-Host "`nDone!" -ForegroundColor Green  
  
$host.UI.RawUI.ForegroundColor = "Gray"  
Write-Host "`nPress any key to continue..."  
  
Read-Host # or pause
```

## Replace Underscore with Hyphen

---

```
Clear-Host; $host.UI.RawUI.ForegroundColor = "Yellow"  
  
# Replace Underscore with Hyphen  
$Var = "Your_String"  
Write-Host "Before: '$Var'"  
$Var = $Var.Replace("_", "-")  
Write-Host "After: '$Var'"  
  
Write-Host "`nDone!" -ForegroundColor Green  
  
$host.UI.RawUI.ForegroundColor = "Gray"  
Write-Host "`nPress any key to continue..."  
Read-Host # or pause
```

## Check Length of String

---

Checks the length of a string and displays whether it is exactly 10 characters, greater than 10, or less than 10.

```
# Define String  
$String = "This is a Test"  
  
Write-Host "Our String: $String`n"  
Write-Host "String Length: $($String.Length)`n"
```



```
# Check the length of the string
if ($String.Length -eq 10) {
    Write-Host "The length of the string is exactly 10 characters."
    # Do something here if the length is 10.
} ElseIf ($String.Length -gt 10) {
    Write-Host "The length of the string is greater than 10 characters."
    # Do something else if the length is greater than 10.
} else {
    Write-Host "The length of the string is less than 10 characters."
    # Handle cases where length is less than 10.
}; pause # or Read-Host
```

## Detect Pattern in String

---

Searches a string for phone numbers matching the pattern 123-123-1234 using a *regular expression*. A regular expression (regex) is a sequence of characters that defines a search pattern. It is used to match, find, and manipulate text in programming. If any matches are found, it displays each phone number; otherwise, it indicates that no phone numbers were found.

```
# Define the string to search for the phone number pattern.
$Data = "Numerical Password: 123-123-1234 line2 line3 987-654-3210 line4"
Write-Host "Our Data: $Data"

# Regular Expression Pattern 123-123-1234
$Pattern = '\d{3}-\d{3}-\d{4}'

# Search for the pattern in the string.
$Matches = [regex]::Matches($Data, $Pattern)

# Check if there are any matches.
if ($Matches.Count -gt 0) {
    ForEach ($Match in $Matches) {
        Write-Host "Found phone number: $($Match.Value)"
    }
} else {
    Write-Host "No phone numbers found."
}

pause # or Read-Host
```



## Check if String is Empty or Null

```
Clear-Host; $host.UI.RawUI.ForegroundColor = "Yellow"

$Data = "String-Here"
Write-Host "Our Data: $Data"

if (-not ([string]::IsNullOrEmpty($Data)))
{
    Write-Host "Data Found"
}

if ([string]::IsNullOrEmpty($Data))
{
    Write-Host "Data Not Found"
}

Write-Host "`nDone!" -ForegroundColor Green

$host.UI.RawUI.ForegroundColor = "Gray"
Write-Host "`nPress any key to continue..."

Read-Host # or pause
```

## Create Formatted Timestamp

Gets the current date and time in the format yyyyMMdd\_HHmm and stores it in the \$Today variable, then displays a completion message in green.

```
Clear-Host; $host.UI.RawUI.ForegroundColor = "Yellow"

$Today = Get-Date -Format "yyyyMMdd_HHmm"; $Today

Write-Host "`nDone!" -ForegroundColor Green

$host.UI.RawUI.ForegroundColor = "Gray"
Write-Host "`nPress any key to continue..."

Read-Host # or pause
```

**See:** Section 24. Formatted Timestamps



## PowerShell: User Operations

---

How to test from the System Account ([PsExec](#)):

```
PsExec64 -S CMD  
PowerShell -ExecutionPolicy Bypass -File "C:\PowerShell\script.ps1"
```

### Return UPN

#### Method 1

---

**Scope:** Admin, System Account, PC, VDI

A User Principal Name (UPN) is the name of a user in an email address-like format used to uniquely identify a user in a network or directory service, such as Microsoft Active Directory (AD). It typically takes the form: username@domain.com. You probably didn't realize you already knew what a UPN was.

```
# Compact Code  
# This assumes there is a static registry key available.  
  
$UPN = (Get-ItemProperty -Path  
'HKLM:\SOFTWARE\Microsoft\Windows\CurrentVersion\Authentication\LogonUI\Session  
Data' -Name 'LoggedOnUser' | Select-Object -ExpandProperty 'LoggedOnUser'  
-ErrorAction 'SilentlyContinue') -Replace '.*\\"', ''  
  
$UPN  
  
Read-Host # or pause
```



## Method 2

---

**Scope:** Admin, System Account, PC

Review this. Try to understand how powerful it is. The *Get-RegistryValue* function gives us the ability to scan through hive keys until we find a specific reg key. There is no need to know exactly how many subkeys there may be. It searches until it finds the value, assuming there is one.

```
function Get-RegistryValue {
    param (
        [string]$RootKey,
        [string]$RootValue
    )

    try {
        # Process all the subkeys.
        $subKeys = Get-ChildItem -Path $RootKey -ErrorAction
        'SilentlyContinue'

        ForEach ($subKey in $subKeys) {
            if ($subKey) {
                Write-Host "Checking key: $($subKey.PSPath)"

                # Get the Registry value from the current subkey.
                $Value = (Get-ItemProperty -Path $subKey.PSPath -ErrorAction
                'SilentlyContinue').$RootValue
                if ($Value) {
                    # Write-Host "Found Value: $value"
                    return $Value
                }

                # Recursively scan deeper subkeys.
                $subKeyValue = Get-RegistryValue -RootKey $subKey.PSPath
                -RootValue $RootValue
                if ($subKeyValue) {
                    return $subKeyValue
                }
            }
        }
    } catch {
        Write-Host "Error scanning key: $RootKey - $_"
    }
}
```



```
# $loggedOnUserSID
$rootKey1 =
"Registry::HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Authentication\LogonUI"

$rootValue1 = 'LoggedOnUserSID'

$loggedOnUserSID = Get-RegistryValue -RootKey $rootKey1 -RootValue $rootValue1

# ConnectionUserUpn
$rootKey2 =
"Registry::HKEY_USERS\$loggedOnUserSID\Software\Microsoft\Office\16.0\Common\ServicesManagerCache\Identities"

$rootValue2 = 'ConnectionUserUpn'

$connectionUserUpn = Get-RegistryValue -RootKey $rootKey2 -RootValue
$rootValue2

Clear-Host
$connectionUserUpn; pause
```

### Method 3

---

**Scope:** Admin, System Account, PC

```
Clear-Host; $host.UI.RawUI.ForegroundColor = "Yellow"

# Open the base Registry key.
$regHive =
[Microsoft.Win32.RegistryKey]::OpenBaseKey([Microsoft.Win32.RegistryHive]::Loca
lMachine, [Microsoft.Win32.RegistryView]::Registry64)

# Define potential Registry paths in priority order.
$sessionPaths = @(
    "SOFTWARE\Microsoft\Windows\CurrentVersion\Authentication\LogonUI\SessionData\2
",
    "SOFTWARE\Microsoft\Windows\CurrentVersion\Authentication\LogonUI\SessionData\1
",
    "SOFTWARE\Microsoft\Windows\CurrentVersion\Authentication\LogonUI\SessionData"
)
```



```
# Initialize a variable for the UPN.  
$UPN = $Null  
  
# Iterate through the paths until a valid UPN is found.  
ForEach ($path in $sessionPaths) {  
    $regSubKey = $regHive.OpenSubKey($path)  
    if ($regSubKey) {  
        $UPN = $regSubKey.GetValue('LoggedOnUser')  
        if ($UPN) { break }  
    }  
}  
  
# Remove domain prefix if UPN is found.  
if ($UPN) {  
    $UPN = $UPN -Replace '.*\\', ''  
}  
  
# Output the result  
$UPN  
  
Write-Host "`nDone!" -ForegroundColor Green  
  
$host.UI.RawUI.ForegroundColor = "Gray"  
Write-Host "`nPress any key to continue..."  
Read-Host
```

## Method 4

---

**Scope:** Admin, System Account, PC, VDI

```
# Define the Registry path and key.  
$regPath =  
"HKLM:\SOFTWARE\Microsoft\Windows\CurrentVersion\Authentication\LogonUI\Session  
Data"  
  
$regKey = "LoggedOnUser"  
  
# Attempt to retrieve the key value.  
try {  
  
    # Find the latest session.  
    $lastSession = (Get-ChildItem -Path $regPath | Sort-Object {  
[int]$_.PSChildName } | Select-Object -Last 1).PSChildName  
  
    # Retrieve the LoggedOnUser value for the latest session.
```



```
$regValue = (Get-ItemProperty -Path "$regPath\$lastSession" -Name $regKey  
-ErrorAction 'Stop') . $regKey  
  
# Check if the value contains a backslash and extract the part after it.  
if ($regValue -Match '\\\\') {  
  
    $regValue = $regValue -Replace '.*\\\\', ''  
  
}  
  
# Output the final value.  
Write-Output "UPN: $regValue"  
  
} catch {  
  
    Write-Output "Error retrieving the Registry value: $_"  
}
```

## NOTE

---

### Other UPN Locations

HKLM\SOFTWARE\Microsoft\Enrollments\781CFF59-6FEF-44D0-B21D-111111111111 | UPN

HKU\User-SID-Here\Software\Microsoft\Office\16.0\Common\LanguageResources\LocalCache  
| RegionalAndLanguageSettingsAccount

HKU\User-SID-Here\Software\Microsoft\Office\16.0\Common\Identity | ADUsername

HKU\User-SID-Here\Software\Microsoft\Windows\CurrentVersion\Explorer\Substrate |  
The-UPN-Name:Substrate



## Return Current Username

How to test from the System Account ([PsExec](#)):

```
PsExec64 -S CMD  
PowerShell -ExecutionPolicy Bypass -File "C:\PowerShell\script.ps1"
```

### Method 1

---

**Scope:** Admin, System Account, PC, VDI

```
# Compact Code  
# This assumes there is a static registry key available.  
# We use -Replace to remove the unwanted characters from our string.  
  
$LoggedOnUser = (Get-ItemProperty -Path  
'HKLM:\SOFTWARE\Microsoft\Windows\CurrentVersion\Authentication\LogonUI\Session  
Data' -Name 'LoggedOnUser' -ErrorAction 'SilentlyContinue' | Select-Object  
-ExpandProperty 'LoggedOnUser') -Replace '.*\\', '' -Replace '@.*', ''  
  
$LoggedOnUser = (Get-ItemProperty -Path  
'HKLM:\SOFTWARE\Microsoft\Windows\CurrentVersion\Authentication\LogonUI\Session  
Data\1' -Name 'LoggedOnUser' -ErrorAction 'SilentlyContinue' | Select-Object  
-ExpandProperty 'LoggedOnUser') -Replace '.*\\', '' -Replace '@.*', ''  
  
$LoggedOnUser; pause
```

### Method 2

---

**Scope:** Admin, System Account, PC

```
# This will return the current user for single user physical machines.  
  
(Get-WmiObject -Class Win32_Process -Filter "Name = 'explorer.exe'").GetOwner()  
| Select-Object -ExpandProperty 'User'
```

Or

```
$LoggedOnUser = (Get-WmiObject -Class Win32_Process -Filter "Name =  
'explorer.exe'").GetOwner() | Select-Object -ExpandProperty 'User'  
$LoggedOnUser
```



## Method 3

### QWINSTA

**Scope:** Admin, System Account, PC, VDI

Clear-Host

```
# Capture the output of QWINSTA.
$qwinstaOutput = & QWINSTA

$currentUserNames = $Null

ForEach ($Line in $qwinstaOutput) { if ($Line -Match "Active") {
    $currentUserNames = $Line;
    break
}
};

# Used to extract specific parts of a string based on a delimiter pattern.
$currentUserNames = ($currentUserNames -Split '\s[2, ]')[1]

$currentUserNames

pause # or Read-Host
```



## Method 4

### QUSER

**Scope:** Admin, System Account, PC, VDI

```
$ErrorActionPreference = 'SilentlyContinue'
$Username = $Null

$basePath =
'HKLM:\SOFTWARE\Microsoft\Windows\CurrentVersion\Authentication\LogonUI\Session
Data'

foreach ($Session in 1..3) {
    try {
        $Username = Get-ItemProperty -Path "$basePath\$Session" -Name
        'LoggedOnSAMUser' | Select-Object -ExpandProperty 'LoggedOnSAMUser'
        if (-not [string]::IsNullOrEmpty($Username)) {
            break
        }
    } catch {
        # Ignore errors for non-existent paths or properties.
    }
}

if ([string]::IsNullOrEmpty($Username))
{
    $quserOutput = QUSER # Verify this exists on your machine.
    $currentUser = $quserOutput | ForEach-Object {
        if ($_.Match '\s+(\S+)\s+(\S+)\s+(\d+)\s+Active\s+') {
            $matches[1]
        }
    }
}

# $currentUser should come from QUSER; if QUSER fails, parse from $Username.
if (-not ($currentUser)) {
    $currentUser = $Username -Split '\\\\' | Select-Object -Last 1
}
Clear-Host
Write-Host "Domain User: $Username"
Write-Host "Parsed: $currentUser"

Write-Host "`n"; pause # or Read-Host
```



## Assessment

'\s+(\S+)\s+(\S+)\s+(\d+)\s+Active\s+'

MATCHES	PURPOSE
\s+	Leading whitespace.
(\S+)	The first non-whitespace string (Ex: first name).
\s+	Whitespace.
(\S+)	The second non-whitespace string (Ex: last name).
\s+	Whitespace.
(\d+)	A sequence of digits (Ex: a user ID).
\s+	Whitespace.
<b>Active</b>	The literal word "Active".
\s+	Trailing whitespace.

## Method 5

QUSER

**Scope:** Current User

```
$quserOutput = QUSER # Verify this exists on your machine.

$currentUser = $quserOutput | ForEach-Object {
    if ($_.Match "^>\s*(\S+)\s+(\S+)\s+(\d+)\s+Active\s+")
        $matches[1]
}
Clear-Host

Write-Host "Current User: $currentUser"
Write-Host "`n"; pause
```



Or

**Scope:** System Account

```
$quserOutput = QUSER 2>$Null

$currentUser | 
    ForEach-Object {
        if ($_.Match '^(\S+)\s+') {
            $matches[1]
        }
    } | Select-Object -First 2

Clear-Host

Write-Host "Current User: $currentUser"

Write-Host "`n"

pause # or Read-Host
```

## Method 6

---

**Scope:** Admin, System Account, PC

```
$ErrorActionPreference = 'SilentlyContinue'

# Initialize Variables
$sessionPaths = 5..1 # Session paths to check in descending order
$uName = ""
$uNameParsed = ""

# Iterate through session paths.
ForEach ($Session in $sessionPaths) {
    try {
        # Attempt to get the username from the Registry.
        $uName = (Get-ItemProperty -Path
        "HKLM:\Software\Microsoft\Windows\CurrentVersion\Authentication\LogonUI\Session
        Data\$Session" -Name 'LoggedOnSAMUser').LoggedOnSAMUser

        $uNameParsed = $uName -Replace '^.*\\', '' | ForEach-Object {
            $_.ToUpper()
        }

        # Exit loop if valid username is found.
        if ($uNameParsed -ne "" -and $uNameParsed -ne "ADMINISTRATOR") {
            break
        }
    }
}
```



```
        }
    }
catch {
    # Ignore errors and continue to the next session.
    continue
};

# Final Check
if ($uNameParsed -eq "" -or $uNameParsed -eq "ADMINISTRATOR") { exit 0 }

Clear-Host

# Output Results
Write-Output "Username: $uName"

Write-Output "Parsed Username: $uNameParsed"

pause # or Read-Host
```



## Method 7

---

**Scope:** Admin, System Account, PC

```
$ErrorActionPreference = 'SilentlyContinue'

# Regular expression pattern is used to match SIDs of two specific formats.
$patternSID = 'S-1-12-1-\d+-\d+-\d+-\d{4,10}$|S-1-5-21-\d+-\d+-\d+-\d{4,7}$'

# Get the profile information matching the SID pattern.
$profileData = Get-ItemProperty 'HKLM:\SOFTWARE\Microsoft\Windows
NT\CurrentVersion\ProfileList\*' |
    Where-Object { $_.PSChildName -Match $patternSID } |
    Select-Object @{'name="Username";expression={$_.ProfileImagePath -Replace
'^(.*/[\\\/])', ''} }

# Iterate through each item and assign the values to variables.
ForEach ($Entry in $profileData) {
    $uName = $Entry.Username
    $uName
}

pause # or Read-Host
```

## Return User SIDs

### Method 1

---

**Scope:** Admin, System Account, PC, VDI

```
# Compact Code
Get-ItemProperty 'HKLM:\SOFTWARE\Microsoft\Windows
NT\CurrentVersion\ProfileList\*' | Where-Object {
    $_.PSChildName -Match
    'S-1-12-1-\d+-\d+-\d+-\d{4,10}$|S-1-5-21-\d+-\d+-\d+-\d{4,7}$' } |
    ForEach-Object { $_.PSChildName };

Pause # or Read-Host
```

Or



**Scope:** Admin, System Account, PC, VDI

```
# Compact Code
Get-ItemProperty 'HKLM:\SOFTWARE\Microsoft\Windows
NT\CurrentVersion\ProfileList\*' | Where-Object {
    $_.PSChildName -Match
    'S-1-12-1-\d+-\d+-\d+-\d{4,10}$|S-1-5-21-\d+-\d+-\d+-\d{4,7}$' } |
Select-Object -ExpandProperty 'PSChildName'
pause # or Read-Host
```

## Method 2

---

**Scope:** Admin, System Account, PC, VDI

Review this. Try to understand how powerful it is. The *Get-RegistryValue* function gives us the ability to scan through hive keys until we find a specific Reg Key. There is no need to know exactly how many subkeys there may be. It searches until it finds the value, assuming there is one.

```
function Get-RegistryValue {
    param (
        [string]$RootKey,
        [string]$RootValue
    )

    try {
        # Process Subkeys
        $subKeys = Get-ChildItem -Path $RootKey -ErrorAction
        'SilentlyContinue'

        ForEach ($subKey in $subKeys) {
            if ($subKey) {
                Write-Host "Checking key: $($subKey.PSPath)"

                # Get the Registry value from the current subkey.
                $Value = (Get-ItemProperty -Path $subKey.PSPath -ErrorAction
                'SilentlyContinue').$RootValue
                if ($Value) {
                    #Write-Host "Found Value: $value"
                    return $Value
                }
            }

            # Recursively scan deeper subkeys.
        }
    }
}
```



```
$subKeyValue = Get-RegistryValue -RootKey $subKey.PSPPath
-RootValue $RootValue
if ($subKeyValue) {
    return $subKeyValue
}
}
}
}
} catch {
    Write-Host "Error scanning key: $RootKey - $_"
}
}

# LoggedOnUserSID
$rootKey1 =
"Registry::HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Authentication\LogonUI"

$rootValue1 = 'LoggedOnUserSID'

$loggedOnUserSID = Get-RegistryValue -RootKey $rootKey1 -RootValue $rootValue1

Clear-Host

$loggedOnUserSID

pause # or Read-Host
```

### Method 3

---

**Scope:** Admin, System Account, PC, VDI

```
$ErrorActionPreference = 'SilentlyContinue'

# Regular expression pattern is used to match SIDs of two specific formats.
$patternSID = 'S-1-12-1-\d+-\d+-\d+-\d{4,10}$|S-1-5-21-\d+-\d+-\d+-\d{4,7}$'

# Get the profile information matching the SID pattern.
$profileData = Get-ItemProperty 'HKLM:\SOFTWARE\Microsoft\Windows
NT\CurrentVersion\ProfileList\*' |
Where-Object { $_.PSChildName -Match $patternSID } |
Select-Object @{name="SID";expression={

    $_.PSChildName
}}
```



```
# Iterate through each item and assign the values to variables.  
ForEach ($Entry in $profileData) {  
    $uSID = $Entry.SID  
    $uSID  
}  
pause # or Read-Host
```

## Method 4

---

**Scope:** Admin, System Account, PC

```
# Get the currently logged-in user name -with domain.  
$USER = (Get-CimInstance -ClassName Win32_ComputerSystem).UserName  
  
# Split the DOMAIN and USER.  
$Domain, $Username = $USER -Split '\\'  
  
# Get the SID of the user by querying Win32_UserAccount.  
$uSID = Get-CimInstance -ClassName Win32_UserAccount | Where-Object { $_.Domain -eq $Domain -and $_.Name -eq $Username } | Select-Object -ExpandProperty 'SID'  
  
# Display the user SID.  
Write-Host "User SID: $uSID"  
pause # or Read-Host
```

## Method 5

---

**Scope:** Admin, System Account, PC

```
$regHive =  
[Microsoft.Win32.RegistryKey]::OpenBaseKey([Microsoft.Win32.RegistryHive]::LocalMachine, [Microsoft.Win32.RegistryView]::Registry64)  
  
$regSubKey =  
$regHive.OpenSubKey("SOFTWARE\Microsoft\Windows\CurrentVersion\Authentication\LogonUI")  
  
$uSID = $regSubKey.GetValue("LastLoggedOnUserSID")  
$uSID  
pause # or Read-Host
```



## Method 6

---

**Scope:** Admin, System Account, PC

```
$regHive =  
[Microsoft.Win32.RegistryKey]::OpenBaseKey([Microsoft.Win32.RegistryHive]::Loca  
lMachine, [Microsoft.Win32.RegistryView]::Registry64)  
  
$regSubKey =  
$regHive.OpenSubKey("SOFTWARE\Microsoft\Windows\CurrentVersion\Authentication\L  
ogonUI")  
  
$uSID = $regSubKey.GetValue("SelectedUserSID")  
  
$uSID  
  
pause # or Read-Host
```



## Assessment

'S-1-12-1-\d+-\d+-\d+-\d{4,10}\\$|S-1-5-21-\d+-\d+-\d+-\d{4,7}\\$'

MATCHES	PURPOSE
<b>S-1-12-1-</b>	This is a literal string that specifies the first part of the SID. "S" represents the SID (Security Identifier) structure, and "1-12-1" represents a specific authority and identifier subauthority.
\d+	Matches one or more digits. This represents the variable parts of the SID, which are numeric. There are three \d+ components that represent the numbers in the SID.
\d{4,10}	This matches a sequence of between 4 and 10 digits. This part represents the last numeric component of the SID and has a length constraint.
\$	The second non-whitespace string (Ex: last name).
	The pipe symbol ( ) acts as an "OR" operator.
<b>S-1-5-21-</b>	This is another literal string that represents a different type of SID format. It specifies another authority and identifier subauthority, in this case, "S-1-5-21".
\d+	Similar to the previous pattern, this matches one or more digits.
\$	The end-of-line anchor ensures the SID ends at this point.



## Return User Domain

### Method 1

**Scope:** Admin, System Account, PC

```
$regPath =
'HKLM:\SOFTWARE\Microsoft\Windows\CurrentVersion\Authentication\LogonUI\Session
Data'

$lastSession = (Get-ChildItem -Path "$regPath" | Sort-Object {
[int]$_.PSChildName } | Select-Object -Last 1).PSChildName

$userData = Get-ItemProperty -Path "$regPath\$lastSession" -Name
'LoggedOnSAMUser' -ErrorAction 'SilentlyContinue'

if ($userData -and $userData.LoggedOnSAMUser) {
    $Domain = $userData.LoggedOnSAMUser -Split '\\\' | Select-Object -First 1
    Write-Output "Domain: $Domain"
} else {
    Write-Output "No logged-in user detected."
}

pause # or Read-Host
```

### Method 2

**Scope:** Admin, System Account, PC

```
$errorActionPreference = 'SilentlyContinue'

# Regular expression pattern is used to match SIDs of two specific formats.
$patternSID = 'S-1-12-1-\d+-\d+-\d+-\d{4,10}$|S-1-5-21-\d+-\d+-\d+-\d{4,7}$'

# Get the profile information matching the SID pattern.
$profileData = Get-ItemProperty 'HKLM:\SOFTWARE\Microsoft\Windows
NT\CurrentVersion\ProfileList\*' |
Where-Object { $_.PSChildName -Match $patternSID } |
Select-Object @{name="SID";expression={
    $_.PSChildName
}}
```



```
# Iterate through each item and assign the values to variables.  
ForEach ($Entry in $profileData) {  
    $uSID = $Entry.SID  
    $uRegPath = "Registry::HKEY_USERS\$($uSID)\Volatile Environment"  
    $uDomain = (Get-ItemProperty -Path "$uRegPath" -Name 'USERDOMAIN'  
    -ErrorAction 'SilentlyContinue').USERDOMAIN  
    $uDomain  
}  
  
pause # or Read-Host
```

## Return User Profile Path

### Method 1

---

**Scope:** Admin, System Account, PC, VDI

```
$ErrorActionPreference = 'SilentlyContinue'  
  
# Regular expression pattern is used to match SIDs of two specific formats.  
$patternSID = 'S-1-12-1-\d+-\d+-\d+-\d{4,10}$|S-1-5-21-\d+-\d+-\d+-\d{4,7}$'  
  
# Get the profile information matching the SID pattern.  
# This can return multiple user info.  
$profileData = Get-ItemProperty 'HKLM:\SOFTWARE\Microsoft\Windows  
NT\CurrentVersion\ProfileList\*' |  
    Where-Object { $_.PSChildName -Match $patternSID } |  
    Select-Object @{'name="Profile";expression={  
        $_.ProfileImagePath  
    }}  
  
# Iterate through each item and assign the values to variables.  
ForEach ($Entry in $profileData) {  
    $uProfile = $Entry.Profile  
    $uProfile  
}  
  
pause # or Read-Host
```



## Method 2

---

**Scope:** Admin, System Account, PC, VDI

```
# Retrieve the SID of the selected user from the registry.  
$selectedUserID = Get-ItemProperty -Path  
"HKLM:\SOFTWARE\Microsoft\Windows\CurrentVersion\Authentication\LogonUI" -Name  
'SelectedUserID' -ErrorAction 'SilentlyContinue' | Select-Object  
-ExpandProperty 'SelectedUserID'  
  
if (-not $selectedUserID) {  
    Write-Error "Could not retrieve the SelectedUserID."  
    exit 1  
}  
  
# Retrieve the profile path for the selected user using the SID.  
$profilePath = Get-ItemProperty -Path "HKLM:\SOFTWARE\Microsoft\Windows  
NT\CurrentVersion\ProfileList\$selectedUserID" -Name 'ProfileImagePath'  
-ErrorAction 'SilentlyContinue' | Select-Object -ExpandProperty  
'ProfileImagePath'  
  
$profilePath  
  
pause # or Read-Host
```

## Return Appdata Folder

---

**Scope:** Admin, System Account, PC, VDI

```
$errorActionPreference = 'SilentlyContinue'  
  
# Regular expression pattern is used to match SIDs of two specific formats.  
$patternSID = 'S-1-12-1-\d+-\d+-\d+-\d{4,10}$|S-1-5-21-\d+-\d+-\d+-\d{4,7}$'  
  
# Get the profile information matching the SID pattern.  
$profileData = Get-ItemProperty 'HKLM:\SOFTWARE\Microsoft\Windows  
NT\CurrentVersion\ProfileList\*' |  
    Where-Object { $_.PSChildName -Match $patternSID } |  
    Select-Object @{name="SID";expression={$_.PSChildName}},  
                  @{name="Profile";expression={$_.ProfileImagePath}}  
}  
  
# Iterate through each item and assign the values to variables.  
ForEach ($Entry in $profileData) {  
    $uSID = $Entry.SID
```



```
$uRegPath = "Registry::HKEY_USERS\$($uSID)\Volatile Environment"
$uAppData = (Get-ItemProperty -Path $uRegPath -Name 'APPDATA' -ErrorAction
'SilentlyContinue').APPDATA
$uAppData
}
```

## Return LocalAppData Folder

---

**Scope:** Admin, System Account, PC, VDI

```
$ErrorActionPreference = 'SilentlyContinue'

# Regular expression pattern is used to match SIDs of two specific formats.
$patternSID = 'S-1-12-1-\d+-\d+-\d+-\d{4,10}$|S-1-5-21-\d+-\d+-\d+-\d{4,7}$'

# Get the profile information matching the SID pattern.
$profileData = Get-ItemProperty 'HKLM:\SOFTWARE\Microsoft\Windows
NT\CurrentVersion\ProfileList\*' |
Where-Object { $_.PSChildName -Match $patternSID } |
Select-Object @{name="SID";expression={$_.PSChildName}},@{name="Profile";expression={$_.ProfileImagePath} }

# Iterate through each item and assign the values to variables.
ForEach ($Entry in $profileData) {

    $uSID = $Entry.SID

    $uRegPath = "Registry::HKEY_USERS\$($uSID)\Volatile Environment"

    $uLocalAppData = (Get-ItemProperty -Path $uRegPath -Name 'LOCALAPPDATA'
-ErrorAction 'SilentlyContinue').LOCALAPPDATA

    $uLocalAppData
}

pause # or Read-Host
```



## Add User to Group

---

**Scope:** Admin, System Account, PC, VDI

Defines a function `Get-RegistryValue` to recursively search through registry subkeys and retrieve a specified value. It first retrieves the logged-on user's SID from the registry, then uses that SID to fetch the User Principal Name (UPN) associated with the user. After acquiring the UPN, the script checks if the user is already a member of the local Administrators group and, if not, adds them to the group. The function handles errors silently and ensures efficient registry querying and group membership management.

```
function Get-RegistryValue {
    param (
        [string]$RootKey,
        [string]$RootValue
    )

    try {
        $subKeys = Get-ChildItem -Path $RootKey -ErrorAction
        'SilentlyContinue'

        ForEach ($subKey in $subKeys) {
            if ($subKey) {
                $Value = (Get-ItemProperty -Path $subKey.PSPath -ErrorAction
                'SilentlyContinue').$RootValue
                if ($Value) {return $Value}
                $subKeyValue = Get-RegistryValue -RootKey $subKey.PSPath
                -RootValue $RootValue
                # Important Bit - It returns the user SID and then the UPN.
                if ($subKeyValue) {return $subKeyValue}
            }
        }
    } catch { Write-Host "There was an error." }
}

# $loggedOnUserSID
$rootKey1 =
"Registry::HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Authentication\LogonUI"

$rootValue1 = 'LoggedOnUserSID'

# Call reg function to acquire the user SID, which will be used for the UPN.
```



```
$loggedOnUserSID = Get-RegistryValue -RootKey $rootKey1 -RootValue $rootValue1

# ConnectionUserUpn
$rootKey2 =
"Registry::HKEY_USERS\$loggedOnUserSID\Software\Microsoft\Office\16.0\Common\ServicesManagerCache\Identities"

$rootValue2 = 'ConnectionUserUpn'
# Call our Registry function to acquire the UPN we need.
$connectionUserUpn = Get-RegistryValue -RootKey $rootKey2 -RootValue
$rootValue2

function Add-LocalGroupMemberSilent($groupName, $Username) {
    $samAccountName = $Username.Split('@')[0]

    $existingMember = Get-LocalGroupMember -Name $groupName -ErrorAction
    'SilentlyContinue' | Where-Object {
        $_.Name -Match "$samAccountName"
    } -ErrorAction 'SilentlyContinue'

    if ($existingMember) {
        Write-Host "TRUE"
        #Remove-LocalGroupMember -Group "$groupName" -Member "$Username"
    } else {
        Write-Host "FALSE"
        Add-LocalGroupMember -Group "$groupName" -Member "$Username"
    }
}

# Call our add to local admin group function.
# We use the dynamically returned UPN here.
Add-LocalGroupMemberSilent Administrators $connectionUserUpn

pause # or Read-Host
```

Or

**Scope:** Admin

```
$Group = 'Administrators'
# The $Member will need to be dynamically retrieved from somewhere.
$Member = "Domain-Here\Username-Here"

try {
```



```
Add-LocalGroupMember -Group $Group -Member $Member -ErrorAction 'Stop'  
}  
  
catch [Microsoft.PowerShell.Commands.MemberExistsException] {  
  
    Write-Warning "$Member already in $Group"  
    #Remove-LocalGroupMember -Group $Group -Member $Member -ErrorAction Stop  
}  
  
pause # or Read-Host
```

## Return Domain Users in the Administrators Group

---

**Scope:** Admin, System Account, PC, VDI

```
# Run the 'net localgroup administrators' command and capture the output.  
$adminGroupMembers = NET LOCALGROUP Administrators  
  
# Filter the members that contain a backslash, which indicates a domain  
account.  
$domainAccounts = $adminGroupMembers | Where-Object { $_ -Match '\\\' }  
  
# Output the domain accounts.  
$domainAccounts  
  
pause # or Read-Host
```

## Detect if User is in Administrators Group

---

### SECURITY IDENTIFIERS WELL KNOWN SID TYPE

---

#### Method1

---

**Scope:** Current User

```
# Returns username - Not reliable when using the System account.  
$User = whoami.exe # Remember security context matters. Commands like this, and  
user environmental variables are not very useful when deploying using the  
system account, or a different user account.
```



```
# Admin Members
$adminMembers = Get-LocalGroupMember -Name 'Administrators' | Select-Object
-ExpandProperty 'Name'

# $adminMembers

# Is User in Admin Group
if ($adminMembers -contains $User) { "$User IS directly in admin group."} else
{ "$User IS NOT directly in admin group."}

pause # or Read-Host
```

## Method 2

---

**Scope:** Admin, System Account, PC, VDI

```
$foundMatch = $false

$regPaths = @();
# Get members of the Administrators group.
$adminGroupMembers = NET LOCALGROUP Administrators

# Filters for domain and local accounts.
$domainAccounts = $adminGroupMembers | Where-Object { $_ -Match '\\\\' }

$localAccounts = $adminGroupMembers | Where-Object { $_ -NoMatch '\\\\' }

# Define Registry Paths
# I do this to account for physical and virtual sessions. You need to
# verify these in your own environment. A better method is to recursively
# scan SessionData and subkeys.
$regPaths =
@("HKLM:\SOFTWARE\Microsoft\Windows\CurrentVersion\Authentication\LogonUI\SessionData\2",
"HKLM:\SOFTWARE\Microsoft\Windows\CurrentVersion\Authentication\LogonUI\SessionData\1",
"HKLM:\SOFTWARE\Microsoft\Windows\CurrentVersion\Authentication\LogonUI\SessionData") ;
```



```
# Scan for a LoggedOnUser key
ForEach ($regPath in $regPaths) {
    $regKey = (Get-ItemProperty -Path "$regPath" -Name 'LoggedOnUser' -ErrorAction 'SilentlyContinue').LoggedOnUser
    if ($regKey) {break}
};

$regKey = 'LoggedOnUser'

# Get the logged-on user from the Registry.
$loggedOnUser = (Get-ItemProperty -Path "$regPath" -Name "$regKey" -ErrorAction 'SilentlyContinue').$regKey

# Extract the part after the backslash if it exists.
if ($loggedOnUser -Match '\\(.*)') { $loggedOnUser = $matches[1] }

# Extract the part before the "@" symbol if it exists.
if ($loggedOnUser -Match '^(.*)@') { $loggedOnUser = $matches[1] }

# Check Domain Accounts
ForEach ($Account in $domainAccounts) {
    if ($Account -Match $loggedOnUser) {
        $foundMatch = $true
        break
    }
};

# Check Local Accounts
ForEach ($Account in $localAccounts) {
    if ($Account -Match $loggedOnUser) {
        $foundMatch = $true
        break
    }
};

# Output
if ($foundMatch) {
    Write-Host "$($loggedOnUser): ADMIN`n"
} else {
    Write-Host "$($loggedOnUser): USER`n"
}
pause # or Read-Host
```



## Detect if User is 'Elevated' as Admin

### Method 1

```
-----  
$isAdmin = ([Security.Principal.WindowsPrincipal]  
[Security.Principal.WindowsIdentity]::GetCurrent()).IsInRole([Security.Principa  
l.WindowsBuiltInRole]::Administrator)  
  
if ($isAdmin) {  
  
    Write-Output "User is an Administrator."  
  
} else {  
  
    Write-Output "User is NOT an Administrator."  
  
}  
  
pause # or Read-Host
```

### Method 2

```
-----  
$elevated = (New-Object  
Security.Principal.WindowsPrincipal([Security.Principal.WindowsIdentity]::GetCurrent())).IsInRole([Security.Principal.WindowsBuiltInRole]::Administrator)  
  
$elevationLevel = if ($elevated) {  
  
    "Administrator"  
  
} else {  
  
    "Standard User"  
}  
  
Write-Output "User is running as: $elevationLevel"  
  
pause # or Read-Host
```



## Return Usernames in Loop

### Method 1

---

**Scope:** Admin, System Account, PC, VDI

Retrieves user SIDs from the registry, specifically from the ProfileList key, and stores them in the \$uSIDs array. Then, for each SID, it checks if a Volatile Environment key exists in the HKEY\_USERS hive; if it does, the script attempts to retrieve the associated USERNAME value and outputs it.

```
# Fast in a physical or virtual environment.
# Local & Domain Accounts

$uSIDs = @();

Get-ItemProperty 'HKLM:\SOFTWARE\Microsoft\Windows
NT\CurrentVersion\ProfileList\*' | Where-Object { $_.PSChildName -ne $Null } |
ForEach-Object {

    $uSIDs += $_.PSChildName

};

ForEach ($uSID in $uSIDs) {
    if (Test-Path "Registry::HKEY_USERS\$uSID\Volatile Environment"
        -ErrorAction 'SilentlyContinue') {

        $uRegistry = "Registry::HKEY_USERS\$uSID\Volatile Environment"

        $uName = (Get-ItemProperty -Path $uRegistry -Name 'USERNAME'
            -ErrorAction 'SilentlyContinue').USERNAME

        $uName

    }
};

pause # or Read-Host
```



## Method 2

---

**Scope:** Admin, System Account, PC, VDI

```
# Fast Physical or Virtual Environment
# Local Accounts Only

Get-WmiObject -Class Win32_UserAccount -Filter "LocalAccount = 'True'" |
ForEach-Object {
    Write-Host "$($_.Name)"

    # add your code here

}

pause # or Read-Host
```

## Method 3

---

**Scope:** Admin, System Account, PC, VDI

```
# Can be slow in a virtual environment.
ForEach ($User in Get-WmiObject Win32_UserAccount | Select-Object -Property
    'Name' ) {

    Write-Host "$($_.Name)"

    # add your code here

}

pause # or Read-Host
```



## Return Usernames, SIDs, LocalAppData

```
Clear-Host

$ErrorActionPreference = 'SilentlyContinue'

# Regular expression pattern is used to match SIDs of two specific formats.
$patternSID = 'S-1-12-1-\d+-\d+-\d+-\d{4,10}$|S-1-5-21-\d+-\d+-\d+-\d{4,7}$'

# Get the profile information matching the SID pattern.
$profileData = Get-ItemProperty 'HKLM:\SOFTWARE\Microsoft\Windows NT\CurrentVersion\ProfileList\*' |
    Where-Object { $_.PSChildName -Match $patternSID } |
    Select-Object @{'name="SID";expression={$_.PSChildName}}, 
        @{'name="Profile";expression={$_.ProfileImagePath}}, 
        @{'name="Username";expression={$_.ProfileImagePath -Replace '^(.*/[\\\/] )', ''}}
    }

# Iterate through each item and assign the values to variables.
ForEach ($Entry in $profileData) {
    $uSID = $Entry.SID

    $uRegPath = "Registry::HKEY_USERS\$($uSID)\Volatile Environment"

    $uDomain = (Get-ItemProperty -Path $uRegPath -Name 'USERDOMAIN' -ErrorAction 'SilentlyContinue').USERDOMAIN

    $uAppData = (Get-ItemProperty -Path $uRegPath -Name 'APPDATA' -ErrorAction 'SilentlyContinue').APPDATA

    $Active = if (-not ($uAppData)) { "Offline" } else { "Online" }

    $uLocalAppData = (Get-ItemProperty -Path $uRegPath -Name 'LOCALAPPDATA' -ErrorAction 'SilentlyContinue').LOCALAPPDATA

    $uProfile = $Entry.Profile

    $uName = $Entry.Username

    $adminsOutput = NET LOCALGROUP Administrators
    $Lines = $adminsOutput -Split "`r`n"
    $Lines = $Lines | ForEach-Object { $_.Trim() }
```



```
# Return elevation status for primary user domain and local member.
$domainMembers = $Lines | Where-Object { $_ -Match "^\w\.-]+\\"[\w\.-]+\$" }
}
$isDomain = $domainMembers | Where-Object { $_ -Match
[regex]::Escape($uName) }
$domainElevation = if ($isDomain) { "ADMIN" } else { "USER" }

$localMembers = $Lines | Where-Object { $_ -Match "^\w\.-]+\$" }
$isLocal = $localMembers | Where-Object { $_ -Match
[regex]::Escape($uName) }
$localElevation = if ($isLocal) { "ADMIN" } else { "USER" }

if ($domainElevation -eq "ADMIN" -or $localElevation -eq "ADMIN")
{$Elevation = "ADMIN"} else {$Elevation = "USER"}

# Return Chrome Version
$chromePath = "C:\Program Files
(x86)\Google\Chrome\Application\chrome.exe"
if (Test-Path $chromePath) {
    $chromeVersion = (Get-Item $chromePath).VersionInfo.FileVersion
}

$chromePath = "C:\Program Files\Google\Chrome\Application\chrome.exe"
if (Test-Path $chromePath) {
    $chromeVersion = (Get-Item $chromePath).VersionInfo.FileVersion
}

# Return SCCM Version
$sccmPath = "C:\Windows\ccmsetup\ccmsetup.exe"
if (Test-Path $sccmPath) {
    $sccmVersion = (Get-Item $sccmPath).VersionInfo.FileVersion
}

# Detect FSLogix
$fslogixPath = "C:\Program Files\FSLogix"
$FsLogix = if (Test-Path $fslogixPath) { $true } else { $false }

# Detect if AzureAD
$dsRegStatus = & DSREGCMD /STATUS
$isAzureADJoined = $dsRegStatus -Match "AzureAdJoined\s*:\s*YES"
$azureAD = if ($isAzureADJoined) { $true } else { $false }

# Return Windows Build Version.
$osInfo = Get-CimInstance -ClassName Win32_OperatingSystem
```



```
$winVersion = $($osInfo.BuildNumber)

# Run the MANAGE-BDE command and capture the output.
$manageBdeOutput = & MANAGE-BDE -PROTECTORS -GET C: 2>&1

$Lines = $manageBdeOutput -Split "`n"
$passwordPattern = "(\d{6}-\d{6}-\d{6}-\d{6}-\d{6}-\d{6}-\d{6}-\d{6})$"
$Passwords = @()

ForEach ($Line in $Lines) {
    if ($Line -Match $passwordPattern) {
        # Extract and clean the password.
        $Password = $Matches[1].Trim()
        # Add the password to the array.
        $Passwords += $Password
    }
}
$commaSeparatedPasswords = $Passwords -Join ","

# Output
Write-Output "-----"
Write-Output "SID: $uSID"
Write-Output "Profile: $uProfile"
Write-Output "AppData: $uAppData"
Write-Output "LocalAppData: $uLocalAppData"
Write-Output "Username: $uName"
Write-Output "Local Elevation: $localElevation"
Write-Output "Domain Elevation: $domainElevation"
Write-Output "Domain: $uDomain"
Write-Output "Registry: HKEY_USERS\$($uSID)"
Write-Output "Active: $Active"
Write-Output "-----"
}

Write-Output "Chrome Version: $chromeVersion"
Write-Output "SCCM Version: $sccmVersion"
Write-Output "Windows Version: $winVersion"
Write-Output "FSLogix: $FsLogix"
Write-Output "AzureAD: $azureAD"
Write-Output "-----"
Write-Output $commaSeparatedPasswords

pause # or Read-Host
```



## Assessment

'(\d{6}-\d{6}-\d{6}-\d{6}-\d{6}-\d{6}-\d{6}-\d{6})\$'

MATCHES	PURPOSE
(\d{6})	\d: Matches any digit (0-9). {6}: Specifies that exactly 6 digits should be matched. This pattern is repeated 8 times with dashes (-) between each set of 6 digits.
\$	The end-of-line anchor ensures the SID ends at this point.

**Example:** 123456-654321-987654-123456-654321-987654-123456-654321

Does that look familiar to you?

**Hint:** Encryption.



## PowerShell: File & Folder Operations

---

How to test from the System Account (**PsExec**):

```
PsExec64 -S CMD  
PowerShell -ExecutionPolicy Bypass -File 'C:\PowerShell\script.ps1'
```

### Copy File

---

```
# Define the paths to the files.  
$sourcePath = 'C:\PowerShell\Test1.txt'  
$destinationPath = 'C:\PowerShell\Test2.txt'  
  
if ([System.IO.File]::exists($sourcePath)) {  
    Copy-Item $sourcePath $destinationPath  
    "Source File ($sourcePath) copied to ($destinationPath)"  
}
```

Or

```
# Define the paths to the files.  
$sourcePath = 'C:\PowerShell\Test1.txt'  
$destinationPath = 'C:\PowerShell\Test2.txt'  
  
if (Test-Path $sourcePath) {  
    Copy-Item $sourcePath $destinationPath  
    "Source File ($sourcePath) copied to ($destinationPath)"  
}
```

### Copy File, If Like

---

```
# Define the paths to the files.  
$sourcePath = 'C:\PowerShell\Test1.txt'  
$destinationPath = 'C:\PowerShell\Test2.txt'  
  
# Check if the source file exists and if its name contains 'Test'.  
if (Test-Path $sourcePath -and ($sourcePath -like '*Test*')) {  
    Copy-Item $sourcePath -Destination $destinationPath  
    Write-Output "Source file '$sourcePath' copied to '$destinationPath'."  
} else {  
    Write-Output "File does not exist or does not contain 'Test' in the name."  
}
```



## Copy File, If Pattern

---

```
# Match Pattern
$sourceFiles = @(
    'C:\PowerShell\Test-1-123-1234-filename.txt',
    'C:\PowerShell\Test-1-12-12-123-filename.txt'
    'C:\PowerShell\Test-2-321-321-2234-filename.txt'
)
$destinationPath = 'C:\PowerShell\Copy'

# Verify Folder
if (-not (Test-Path $destinationPath)) {
    Write-Output "Destination directory does not exist. Creating:
    $destinationPath`n"
    New-Item -Path $destinationPath -ItemType Directory -Force
}

# Define the regex pattern for the desired structure.
$pattern = '.*\b\d-\d{3}-\d{3}-\d{4}\b.*'

# Iterate through each file.
foreach ($sourcePath in $sourceFiles) {
    try {
        # Extract the filename from the path.
        $fileName = [System.IO.Path]::GetFileName($sourcePath)
        Write-Output "Processing file: $fileName"

        # Check if the file exists and matches the pattern.
        if ((Test-Path $sourcePath) -and ($fileName -Match $pattern)) {
            Copy-Item -Path $sourcePath -Destination $destinationPath -Force
            Write-Output "Copied '$fileName' to '$destinationPath'.`n"
        } else {
            Write-Output "File '$fileName' does not exist or does not match
            the pattern.`n"
        }
    } catch {
        Write-Output "Error processing file '$sourcePath': $_"
    }
}

Write-Output "Done!"
pause
```



## Assessment

'.\*\b\d-\d{3}-\d{3}-\d{4}\b.\*'

MATCHES	PURPOSE
.*	. matches any character except a newline. * means zero or more of the preceding element. Together, .* matches any sequence of characters (including an empty sequence).
\b	This is a word boundary anchor. It asserts a position where a word character is not followed or preceded by another word character. It ensures that the pattern matches a whole word.
\d	This matches any digit (equivalent to [0-9]).
-	This matches the hyphen character literally.
\d{3}	\d matches any digit. {3} specifies exactly three occurrences of the preceding element. Together, \d{3} matches exactly three digits.
\d{4}	Similar to \d{3}, but matches exactly four digits.
\b	Another word boundary anchor, ensuring the pattern matches a whole word.
.*	As explained earlier, matches any sequence of characters (including an empty sequence).

Putting it all together, the pattern `.*\b\d-\d{3}-\d{3}-\d{4}\b.*` matches any string that contains a sequence of characters followed by a word boundary, a digit, a hyphen, three digits, another hyphen, three more digits, another hyphen, four digits, and another word boundary, followed by any sequence of characters.

This pattern is designed to match a phone number format like X-XXX-XXX-XXXX where X is a digit, ensuring that the phone number is a whole word within the string.



## Delete File

```
# Define the path to the file.  
$filePath = 'C:\PowerShell\File.txt'  
  
if ([System.IO.File]::exists($filePath)) {  
    Remove-Item $filePath | Out-Null  
}  
  
pause # or Read-Host
```

Or

```
# Define the path to the file.  
$filePath = 'C:\PowerShell\File.txt'  
if (Test-Path $filePath) {  
    Remove-Item $filePath | Out-Null  
}  
  
pause # or Read-Host
```

## Delete File with Force

```
# Define the path to the file.  
$filePath = 'C:\PowerShell\File.txt'  
  
# Check if the file exists.  
if (Test-Path $filePath) {  
    # Delete the file with force.  
    Remove-Item -Path $filePath -Force  
    Write-Host "File deleted: $filePath"  
} else {  
    Write-Host "File does not exist: $filePath"  
}  
  
pause # or Read-Host
```



## Set Current Directory Working Path

```
Set-Location -Path $PSScriptRoot # Set-Location .\
```

## Read Contents of File

```
# Define the path to the file.
$filePath = 'C:\PowerShell\File.txt'

# Check if the file exists.
if (Test-Path "$filePath") {
    # Read the contents of the file.
    $contents = Get-Content -Path "$filePath"
    Write-Host $contents
} else {
    Write-Host "File not found: $filePath"
}

pause # or Read-Host
```

## Trim WhiteSpace from File Contents

```
# -----
# LAB SETUP
MD C:\PowerShell | Out-Null
@"

Numerical Password:
line2
line3
line4
"@ | Set-Content -Path "C:\PowerShell\File.txt"
# You can check File.txt. It will have extra spaces.
Clear-Host
# -----

Clear-Host

# Define the path to the file.
$filePath = 'C:\PowerShell\File.txt'

# Read content before trimming.
$originalContent = Get-Content -Path "$filePath"
```



```
# Display original content before trimming.
Write-Host "Original Content:"
$originalContent
Write-Host "`n"

# Trim each line and update the file.
$trimFile = $originalContent | % { $_.Trim() } | Set-Content -Path "$filePath"

# Read content after trimming.
$trimmedContent = Get-Content -Path "$filePath"

# Display trimmed content
Write-Host "Trimmed Content:"
$trimmedContent

pause # or Read-Host
```

## Read File into Array

---

```
# -----
# LAB SETUP
Md C:\PowerShell | Out-Null
@"
Numerical Password:
line2
line3
line4
"@ | Set-Content -Path "C:\PowerShell\File.txt"
Clear-Host
# -----

$filePath = 'C:\PowerShell\File.txt'

[string[]]$Array = Get-Content -Path "$filePath"

ForEach($Item in $Array)
{
    Write-Host $Item
}

pause # or Read-Host
```



## Create Folder

---

```
if (-not (Test-Path "C:\Setup")) {  
    New-Item -Path "C:\Setup" -ItemType Directory | Out-Null  
}  
pause # or Read-Host
```

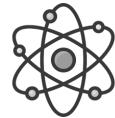
## Detect if Folder is Empty

---

```
# -----  
# LAB SETUP  
Md C:\PowerShell | Out-Null  
@"  
Hello!  
"@ | Set-Content -Path 'C:\PowerShell\File.txt'  
Clear-Host  
# -----  
  
# Compact Code  
$FolderPath = 'C:\PowerShell'  
(Test-Path $FolderPath) -and ((Get-ChildItem -Path $FolderPath).Count -eq 0)
```

Or

```
# Define the path to the folder.  
$FolderPath = 'C:\PowerShell'  
  
# Check if the folder exists.  
if (Test-Path $FolderPath) {  
    # Get the contents of the folder.  
    $folderContents = Get-ChildItem -Path $FolderPath  
  
    # Return True if the folder is empty, else False  
    if ($folderContents.Count -eq 0) {  
        $true # The folder is empty.  
    } else {  
        $false # The folder is not empty.  
    }  
} else {  
    $false # The folder does not exist.  
}  
pause # or Read-Host
```



## Detect if Folder Exists

---

```
# -----
# LAB SETUP
Md C:\PowerShell | Out-Null
Clear-Host
# -----

# Compact Code
Test-Path 'C:\PowerShell' ; pause # or Read-Host
```

Or

```
# Define the path to the folder.
$FolderPath = 'C:\PowerShell'

# Check if the directory exists.
if (Test-Path $FolderPath) {
    $true # The folder exists.
} else {
    $false # The folder does not exist.
}
pause # or Read-Host
```

## Detect, Create or Delete Folder

---

```
# Define the path to the folder.
$FolderPath = 'C:\TestFolder'
if (Test-Path $FolderPath) {
    $(New-Object -ComObject "WScript.Shell").Popup("That folder does
exists!",0,"Folder Path",0)
    Remove-Item -Path $FolderPath
    $(New-Object -ComObject "WScript.Shell").Popup("The folder was
deleted!",0,"Folder Path",0)
}

else {
    $(New-Object -ComObject "WScript.Shell").Popup("That folder does not
exist!",0,"Folder Path",0)

    New-Item -Path $FolderPath -ItemType Directory
    $(New-Object -ComObject "WScript.Shell").Popup("The folder was
created!",0,"Folder Path",0)
}
pause # or Read-Host
```



## Further Testing

```
Get-ChildItem -Path 'C:\TestFolder' -File -Recurse | Where-Object {$_._DirectoryName -NotMatch 'FolderToExclude' -and ($_.LastWriteTime -lt (Get-Date).AddDays(+1))} | Remove-Item
```

## Return File Version using Get-Package

```
-----  
  
(Get-Package -Name "Microsoft Edge*" | Select-Object Version -ExpandProperty 'Version')[0]  
  
pause # or Read-Host
```

## Return File Version using FileVersionInfo/VersionInfo

```
-----  
  
# Compact Code  
$fileVersion =  
($([System.Diagnostics.FileVersionInfo]::GetVersionInfo("C:\Windows\notepad.exe") .FileVersion) -Split ' ')[0]; $fileVersion
```

Or

```
$appPath = "C:\Windows\notepad.exe"  
  
(Get-Item $appPath).VersionInfo.FileVersion | % {$_.Replace(' ', '')} | % {$_.Replace(',', '.')}  
  
(Get-Item $appPath).VersionInfo.FileVersionRaw  
(Get-Item $appPath).VersionInfo.FileVersion  
(Get-Item $appPath).VersionInfo.ProductVersion
```

```
pause # or Read-Host
```

## NOTE

Feature	.FileVersionInfo	.VersionInfo (or .Version)
Applies to	Files (EXE, DLL, etc.)	PowerShell commands (cmdlets, scripts, executables)
Returns	Detailed file version info	Version of the command (if available)



## Verify MD5 on File

### Return MD5 Hash

---

```
# Define the file name to return MD5 for.  
$file_Md5 = Get-FileHash 'C:\Windows\notepad.exe' -Algorithm MD5 |  
Select-Object -ExpandProperty 'Hash'  
  
$file_Md5  
  
pause # or Read-Host
```

### Compare MD5 Hash

---

```
# MD5 we're using for comparison.  
$file_Md5 = 'FF3E29FDFAFA0E9030E2FCD71489D41D'  
  
# Define the file name to return MD5.  
$check_Md5 = Get-FileHash 'C:\Windows\notepad.exe' -Algorithm MD5 |  
Select-Object -ExpandProperty 'Hash'  
  
if ($check_Md5 -ne $file_Md5) {  
    Write-Host "File MD5 Failed! `n"  
    Start-Sleep 2  
    Write-Host "Exiting..."  
    Start-Sleep 2  
    exit  
  
} else {  
  
    Write-Host "File MD5 Verified! `n"  
    Start-Sleep 2  
  
}  
  
pause # or Read-Host
```

### NOTE

---

An MD5 hash is a unique fixed-length string (a 128-bit hexadecimal value) that represents the contents of a file. It is commonly used to verify file integrity—if a file is altered, its MD5 hash will change.



## Unblock Downloaded Files

---

```
# Define the path to the current user Downloads folder.  
$downloadsPath = Join-Path $env:USERPROFILE "Downloads"  
  
# Get a list of files in the Downloads folder.  
$files = Get-ChildItem $downloadsPath -File  
$files  
  
# Iterate through each file in the list.  
ForEach ($file in $files) {  
    $retFile = $file.FullName  
  
    # Check if the file path is not null.  
    if ($retFile -ne $Null) {  
        Write-Host "Unblocking file: $retFile"  
        Unblock-File -LiteralPath $retFile -Verbose  
    } else {  
        Write-Host "File not found: $retFile"  
    }  
  
    # Reset Variable  
    $retFile = ""  
    Write-Host "`n"  
}  
  
# Reset the list and file variables.  
$files = $Null  
$file = $Null  
$retFile = $Null  
  
pause # or Read-Host
```



## PowerShell: Text File Operations

---

How to test from the System Account (**PsExec**):

```
PsExec64 -S CMD  
PowerShell -ExecutionPolicy Bypass -File 'C:\PowerShell\script.ps1'
```

### Create Text File

---

```
$createFile | Out-File 'C:\PowerShell\File.txt'
```

### Write to Text File

---

```
$addTextToFile = 'This is line 1'  
$addTextToFile | Out-File 'C:\PowerShell\File.txt'
```

### Append to Text File

---

```
$addTextToFile = 'This is another line'  
$addTextToFile | Out-File 'C:\PowerShell\File.txt' -Append
```

### Delete Text File

---

```
$textFilePath = 'C:\PowerShell\File.txt'  
Remove-Item "$textFilePath" -Force -ErrorAction 'SilentlyContinue'
```

### Read from Text File

---

```
$textFileContent = Get-Content 'C:\PowerShell\File.txt'  
$textFileContent
```

Or

```
$textFilePath = 'C:\PowerShell\File.txt'
```



```
if (Test-Path "$textFilePath") {
    $textContent = Get-Content "$textFilePath"
    $textContent
}

Write-Host "`nDone!" -ForegroundColor Green
Write-Host "`nPress any key to continue..."
Read-Host
```

Or

```
$textContent = Get-Content 'C:\PowerShell\File.txt'

$(New-Object -ComObject "WScript.Shell").Popup($textContent,0,"Contents of
file",0); pause
```

## Read from Text File, Line by Line

---

```
# -----
# LAB SETUP
$PowerShellDir = "C:\PowerShell"
if (-not (Test-Path $PowerShellDir)) {
    New-Item -Path $PowerShellDir -ItemType Directory
}

$textFilePath = 'C:\PowerShell\File.txt'
# Create the text file with the specified content
$content = @"
Eddie Jackson
Computer-Lab1
TYNEKJG-12
"@
Set-Content -Path $textFilePath -Value $content
Clear-Host
# -----

$textFilePath = 'C:\PowerShell\File.txt'

# Check if the file exists and read its content line by line.
if (Test-Path "$textFilePath") {

    $textContent = Get-Content "$textFilePath"
    Write-Host "Reading file line by line..." ; Start-Sleep 2
    $counter = 1
```



```
foreach ($line in $textFileContent) {  
    Write-Output "Line$($counter): $line" ; Start-Sleep 2  
    $counter++  
}  
} else {  
  
    Write-Output "The file does not exist."  
}  
  
Write-Host "`nDone!" ; Start-Sleep 2  
Write-Host "`nPress any key to continue..."  
Read-Host # or pause
```



## Assessment

### Concepts and Algorithms

#### File I/O

Uses Test-Path to check for file existence.  
Uses Get-Content to read file contents.

#### Looping and Conditional Statements

If statement checks if the file exists.  
Foreach loop iterates through each line of the file.

#### Time and Delays

The Start-Sleep cmdlet introduces delays, enhancing user readability.

#### User Interaction

Write-Host and Write-Output display messages to the user.  
Read-Host waits for user input before closing.

#### Counter Management

A counter variable is used to keep track of line numbers.



## Rename Text Files, Replace Underscore

---

```
# Compact Code
$dir = 'C:\PowerShell'
Get-ChildItem $dir -Filter "file_*.txt" | ForEach-Object {
    $newName = $_.Name -Replace "_" , "-"
    Rename-Item $_.FullName -NewName $newName -ErrorAction 'SilentlyContinue'
}; pause
```

Or

```
# -----
# LAB SETUP
$PowerShellDir = "C:\PowerShell"
if (-not (Test-Path $PowerShellDir)) {
    New-Item -Path $PowerShellDir -ItemType Directory
}

# Create additional files with underscores in their names
$files = @('file_one.txt', 'file_two.txt', 'file_three.txt')
foreach ($file in $files) {
    $filePath = Join-Path -Path $PowerShellDir -ChildPath $file
    Set-Content -Path $filePath -Value "Sample content for $file"
}
Clear-Host
# -----

# Define the path to the folder. This is where our text files are.
$PowerShellDir = "C:\PowerShell"

Write-Host "Current file names: file_one.txt, file_one.txt, file_one.txt`n";
Start-Sleep 2

Write-Host "Replacing underscores in file names..." ; Start-Sleep 2

# Get all files matching the pattern "file_*.txt" in C:\PowerShell.
Get-ChildItem -Path $PowerShellDir -Filter "file_*.txt" | ForEach-Object {

    # Create the new name by replacing the underscore with a hyphen.
    $newName = $_.Name -Replace "_" , "-"

    # Rename the file.
    Rename-Item -Path $_.FullName -NewName $newName -ErrorAction
    'SilentlyContinue'
```



```
Write-Host "Renamed: $($_.Name) to $newName" ; Start-Sleep 2
}

Write-Host "`nDone!" ; Start-Sleep 2

Write-Host "`nPress any key to continue..."
```

Read-Host # or pause



## Assessment

### Concepts and Algorithms

#### File I/O

Uses Get-ChildItem to list files and Rename-Item to rename files.

#### Timers and Delays

The Timer function introduces delays using Start-Sleep, enhancing user readability.

#### User Interaction

Write-Host displays messages to the user.

Read-Host waits for user input before closing.

#### String Manipulation

The -Replace operator is used to replace underscores with hyphens in file names.

#### Error Handling

The -ErrorAction 'SilentlyContinue' parameter ensures that errors during renaming are handled silently without interrupting the script.



## Sort Text File Ascending/Descending

---

```
# Compact Code
$unsortedFile = 'C:\PowerShell\Unsorted.txt'
$sortedFile = 'C:\PowerShell\Sorted.txt'
Get-Content $unsortedFile | Sort-Object | Set-Content $sortedFile
Get-Content $sortedFile
pause
```

Or

```
# -----
# LAB SETUP
$PowerShellDir = "C:\PowerShell"
if (-not (Test-Path $PowerShellDir)) {
    New-Item -Path $PowerShellDir -ItemType Directory
}

# Create Unsorted.txt
@"
B
D
A
F
E
C
"@ | Set-Content -Path "$PowerShellDir\Unsorted.txt"
Clear-Host
# -----

# Define the paths to the files.
$unsortedFile = 'C:\PowerShell\Unsorted.txt'
$sortedFile = 'C:\PowerShell\Sorted.txt'

# This is the timer function to simulate delays.
function Timer {

    param([int]$seconds = 2)
    Start-Sleep -Seconds $seconds
}

# Show Unsorted Content
Write-Host "Unsorted Content:" -ForegroundColor Cyan
Timer 2
Get-Content $unsortedFile
```



```
Write-Host ""

# Sort the content into Sorted.txt - ascending order.
Write-Host "Running our SORT command..." -ForegroundColor Yellow
Timer 2
Get-Content $unsortedFile | Sort-Object | Set-Content -Path $sortedFile
Write-Host ""

# Show Sorted Content
Write-Host "Sorted Content:" -ForegroundColor Cyan
Timer 2

Get-Content $sortedFile

Write-Host "`nDone!" -ForegroundColor Green
Write-Host "`nPress any key to continue..."; Read-Host
```



## Assessment

### Concepts and Algorithms

#### File I/O

Uses Get-Content to read file contents and Set-Content to write to files.

#### Looping and Conditional Statements

ForEach-Object processes each line to trim whitespace and remove empty lines.

#### Timers and Delays

The Timer function introduces delays using Start-Sleep, enhancing user readability.

#### User Interaction

Write-Host displays messages to the user.

Read-Host waits for user input before closing.

#### Sorting and Removing Duplicates

Sort-Object sorts the lines of the file in ascending order.



## Merge Two Text Files

---

```
# Compact Code
$file1 = 'C:\PowerShell\File1.txt'; $file2 = 'C:\PowerShell\File2.txt'
$combinedFile = 'C:\PowerShell\Combined.txt'

Get-Content $file1, $file2 | Set-Content $combinedFile
Get-Content $combinedFile
pause # or Read-Host
```

Or

```
# -----
# LAB SETUP
$PowerShellDir = "C:\PowerShell"
if (-not (Test-Path $PowerShellDir)) {
    New-Item -Path $PowerShellDir -ItemType Directory
}

# File1.txt
@"
Line1
Line2
Line3
"@ | Set-Content -Path "$PowerShellDir\File1.txt"

# File2.txt
@"
Line4
Line5
Line6
"@ | Set-Content -Path "$PowerShellDir\File2.txt"
Clear-Host
# -----

# Define the paths to the files.
$file1 = 'C:\PowerShell\File1.txt'
$file2 = 'C:\PowerShell\File2.txt'
$combinedFile = 'C:\PowerShell\Combined.txt'

# Define a timer function -similar to :TIMER in Batch.
function Timer {

    param([int]$seconds = 2)
    Start-Sleep -Seconds $seconds
```



```
}
```

```
# Display the contents of File1.txt.
Write-Host "Content of File1:" -ForegroundColor Cyan
Get-Content $file1
Timer 2
```

```
# Display the contents of File2.txt.
Write-Host "`nContent of File2:" -ForegroundColor Cyan
Get-Content $file2
Timer 2
```

```
# Inform about the merging process.
Write-Host "`nNow let's merge files into Combined.txt..." -ForegroundColor Yellow
Timer 2
```

```
# Merge the contents of File1.txt and File2.txt into Combined.txt.
Get-Content $file1, $file2 | Set-Content -Path "$combinedFile"
```

```
# Confirmation
Write-Host "`nFiles have been merged!" -ForegroundColor Green
Timer 2
```

```
# Show the content of Combined.txt.
Write-Host "`nShowing content of Combined.txt:" -ForegroundColor Cyan
Get-Content $combinedFile
Timer 2
```

```
Write-Host "`nDone!" -ForegroundColor Green
```

```
Write-Host "`nPress any key to continue..."
Read-Host # or pause
```



## Assessment

### Concepts and Algorithms

#### Timer Function

Defines a Timer function that uses Start-Sleep to introduce delays, mimicking the behavior of the TIMER command in batch files. This enhances user readability by adding pauses between actions.

#### File I/O

Uses Get-Content to read the contents of File1.txt and File2.txt to display them in the console.

Uses Set-Content to write the combined contents of File1.txt and File2.txt into a new file Combined.txt.

#### Looping and Conditional Statements

No explicit loops or conditional statements are used in the script, but Get-Content is used to read and display file content in sequence.

#### User Interaction

Uses Write-Host to display informative messages to the user throughout the script, such as indicating the start of file merging and completion.

Read-Host waits for user input before closing.



## Merge Two Text Files, Remove Dups

```
-----  
# -----  
# LAB SETUP  
$PowerShellDir = "C:\PowerShell"  
if (-not (Test-Path $PowerShellDir)) {  
    New-Item -Path $PowerShellDir -ItemType Directory  
}  
  
# Create File1.txt  
@"  
Line1  
Line2  
Line3  
"@ | Set-Content -Path "$PowerShellDir\File1.txt"  
  
# Create File2.txt  
@"  
Line4  
Line2  
Line5  
Line6  
"@ | Set-Content -Path "$PowerShellDir\File2.txt"  
  
Clear-Host  
# -----  
  
# Define File Paths  
$file1 = 'C:\PowerShell\File1.txt'  
$file2 = 'C:\PowerShell\File2.txt'  
$combinedFile = 'C:\PowerShell\Combined.txt'  
$outputFile = 'C:\PowerShell\RemovedDuplicates.txt'  
  
# Timer Function  
function Timer {  
  
    param([int]$seconds = 2)  
    Start-Sleep -Seconds $seconds  
  
}  
  
# Show the contents of File1.  
Write-Host "Content of File1:" -ForegroundColor Cyan  
Timer 2  
Get-Content $file1  
Write-Host ""
```



```
# Show the contents of File2.
Write-Host "Content of File2:" -ForegroundColor Cyan
Timer 2
Get-Content $file2
Write-Host ""

# Merge File1.txt and File2.txt into Combined.txt.
Write-Host "Now let's merge files into Combined.txt...`n" -ForegroundColor Yellow
Timer 2
Get-Content $file1, $file2 | Set-Content -Path $combinedFile
Write-Host "Files have been merged!`n" -ForegroundColor Green
Timer 2

# Show the combined content.
Write-Host "Content of Combined.txt:" -ForegroundColor Cyan
Get-Content $combinedFile
Write-Host ""

# Remove the duplicates.
Write-Host "There are duplicate entries.`n" -ForegroundColor Yellow
Timer 4
Write-Host "We want to remove any duplicates.`n" -ForegroundColor Yellow
Timer 2
Write-Host "Removing duplicates now..." -ForegroundColor Yellow
Timer 2

# Remove duplicates and write to the output file.
Get-Content $combinedFile |
    Sort-Object -Unique | ForEach-Object { $_.Trim() } | Where-Object { $_ -ne "" } |
        Set-Content -Path $outputFile

Write-Host "`nDuplicates have been removed!`n" -ForegroundColor Green
Timer 4

# Verify the result.
Write-Host "Let's verify...`n" -ForegroundColor Yellow
Timer 2
Write-Host "Content of RemovedDuplicates.txt:" -ForegroundColor Cyan
Get-Content $outputFile
Timer 2

Write-Host "`nDone!" -ForegroundColor Green
Write-Host "`nPress any key to continue..."
Read-Host # or pause
```



## Assessment

### Concepts and Algorithms

#### File I/O

Uses Get-Content to read file contents and Set-Content to write to files.

#### Looping and Conditional Statements

ForEach-Object processes each line to trim whitespace and remove empty lines.

#### Timers and Delays

The Timer function introduces delays using Start-Sleep, enhancing user readability.

#### User Interaction

Write-Host displays messages to the user.

Read-Host waits for user input before closing.

#### Sorting and Removing Duplicates

Sort-Object -Unique sorts the lines and removes duplicates.

Where-Object { \$\_. -ne "" } filters out empty lines.



## Return Specific Line in Text File

---

```
# Compact Code
$filePath = 'C:\PowerShell\Text.dat'; $Line=2; $count=1
Get-Content $filePath | ForEach-Object { if ($count -eq $Line) {
$returnLine=$_ }; $count++ }
$returnLine; pause
```

Or

```
# -----
# LAB SETUP
$PowerShellDir = "C:\PowerShell"
if (-not (Test-Path $PowerShellDir)) {
    New-Item -Path $PowerShellDir -ItemType Directory
}

# Create Text.dat
@"
$env:USERNAME
9F1AE7B34381DFD1111
$env:COMPUTERNAME
"@ | Set-Content -Path "$PowerShellDir\Text.dat"

Clear-Host
# -----

# Define the path to the file.
$filePath = 'C:\PowerShell\Text.dat'
$count = 1
$returnLine = $Null

# Timer Function
function Timer {
    param([int]$seconds = 2)
    Start-Sleep -Seconds $seconds
}

# Display all lines in the file.
Write-Host "Showing each line in $filePath..." -ForegroundColor Cyan
Timer 2

Get-Content $filePath | ForEach-Object {

    if ($count -eq 2) { $returnLine = $_ }
```



```
Write-Host "$count $_"
Timer 2
$count++
}

Write-Host ""

# Display a specific line.
Write-Host "Showing a specific line:" -ForegroundColor Cyan
Timer 2

Write-Host "Line 2: $returnLine"
Timer 2

Write-Host "`nDone!" -ForegroundColor Green

Write-Host "`nPress any key to continue..."
Read-Host
```



## Assessment

### Concepts and Algorithms

#### File I/O:

The script uses Get-Content to read the contents of a file line by line.

#### Looping and Conditional Statements:

ForEach-Object is used to loop through each line of the file.

An if statement checks if the current line number is 2 and stores the line content.

#### Timers and Delays:

The Timer function introduces delays using Start-Sleep, which is useful for creating pauses in script execution.

#### User Interaction:

Write-Host is used to display messages to the user.

[System.Console]::ReadKey(\$true) waits for user input before closing, ensuring the user has time to read the output.

#### Variable Management:

Variables are used to store file paths, counters, and specific line content for later use.



## PowerShell: Process Operations

---

How to test from the System Account (**PsExec**):

```
PsExec64 -S CMD  
PowerShell -ExecutionPolicy Bypass -File 'C:\PowerShell\script.ps1'
```

### Start Process

---

```
Start-Process "notepad" # -Wait -WindowStyle Maximized
```

### Invoke-Expression

---

**Invoke-Expression (iex)** executes a string as a PowerShell command. This is useful when dynamically constructing or modifying a command that needs to be executed at runtime. Since PowerShell treats strings as text by default, Invoke-Expression explicitly tells it to interpret the string as code and run it. However, it should be used with caution, as executing untrusted input can pose security risks.

```
$command = "Get-Process"  
Invoke-Expression $command # Runs Get-Proces
```

### ScriptBlock

---

A **ScriptBlock** is a collection of PowerShell commands enclosed in {} that can be stored, executed, and passed around like an object. It is useful for deferred execution and reusable code.

```
$script = { Get-Date }  
& $script # Executes the script block  
  
pause
```



## Start-Job

---

A PowerShell Job runs a command or script asynchronously in the background (think, parallel processing), allowing the main script to continue executing without waiting for the job to finish. This enables powerful and dynamic scripting by running multiple processes or commands simultaneously, improving efficiency and performance.

```
Start-Job -ScriptBlock { Start-Sleep 5; "Job Done!" }
```

Or

```
# Start a background job.  
$job = Start-Job -ScriptBlock { Start-Sleep -Seconds 5; "Job Completed!" }  
  
# Continue processing while the job runs.  
Write-Host "Job is running... main script continues."  
  
# Wait for the job to complete and get results.  
$job | Wait-Job | Receive-Job  
  
# Clean up the job.  
Remove-Job $job  
  
pause
```

## Stop Process: Graceful

---

```
$Notepad = Get-Process "notepad" -ErrorAction 'SilentlyContinue'  
$Notepad.CloseMainWindow() # Graceful Close
```

## Stop Process: Force

---

```
$Notepad = Get-Process "notepad" -ErrorAction 'SilentlyContinue'  
$Notepad | Stop-Process -Force # Forced Close
```

Or

```
Stop-Process -Name "notepad" -Force
```



## Detect if Process is Running

---

```
$Notepad = Get-Process "notepad" -ErrorAction 'SilentlyContinue'

if ($Notepad) {
    $true
}
else {
    $false
}
pause
```

## Change Process Base Priority

---

**Process Base Priority** is the initial priority level assigned to a process when it is created in the Windows operating system. It determines the scheduling priority of the process and influences how much CPU time the process is allocated relative to other processes. This priority affects how the system's scheduler decides which process to execute next.

```
# Compact Code
$Processes = Get-Process -Name "notepad" -ErrorAction 'SilentlyContinue'
if ($Processes) { $Processes | ForEach-Object { $_.PriorityClass = 'High' } }

Or

# Get all Notepad processes.
$Processes = Get-Process -Name "notepad" -ErrorAction 'SilentlyContinue'

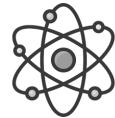
# Check if there are any Notepad processes running.
if ($Processes) {
    ForEach ($process in $Processes) {
        try {
            # Set the base priority to High. Or: Read-time, Above normal,
            # normal, below normal, others.
            $process.PriorityClass =
                [System.Diagnostics.ProcessPriorityClass]::High

            Write-Host "Set priority to High for process ID $($process.Id)
($($process.ProcessName))}"
        catch {
            Write-Host "Failed to set priority for process ID $($process.Id)"
```



```
($(($process.ProcessName)): $_"
    }
}
} else {
    Write-Host "No Notepad processes found."
}

Read-Host # or pause
```



## PowerShell: Registry Operations

---

How to test from the System Account (**PsExec**):

```
PsExec64 -S CMD
PowerShell -ExecutionPolicy Bypass -File 'C:\PowerShell\script.ps1'
```

### Add Reg Key

---

```
# Create a new PSDrive for HKEY_USERS.
if (-not (Get-PSDrive -Name HKU -PSProvider Registry -ErrorAction
    'SilentlyContinue')) {
    New-PSDrive -Name HKU -PSProvider Registry -Root HKEY_USERS | Out-Null
}

# Compact Code
$regHive =
[Microsoft.Win32.RegistryKey]::OpenBaseKey([Microsoft.Win32.RegistryHive]::Curr
entUser, [Microsoft.Win32.RegistryView]::Registry64)

$regHive.CreateSubKey("SOFTWARE\Test").SetValue("myKey", "1",
[Microsoft.Win32.RegistryValueKind]::String)

$regHive.Close()
pause # or Read-Host
```

Or

```
# Create a new PSDrive for HKEY_USERS.
if (-not (Get-PSDrive -Name HKU -PSProvider Registry -ErrorAction
    'SilentlyContinue')) {
    New-PSDrive -Name HKU -PSProvider Registry -Root HKEY_USERS | Out-Null
}

# Define the Registry path and value details.
$regPath = "SOFTWARE\Test"
$regKey = "myKey"
$regValue = "1"

# Open the Registry hive for 64-bit view.
```



```
$regHive =
[Microsoft.Win32.RegistryKey]::OpenBaseKey([Microsoft.Win32.RegistryHive]::CurrentUser, [Microsoft.Win32.RegistryView]::Registry64)
# Create or open the Registry key.
$regSubKey = $regHive.CreateSubKey($regPath)

# Set the value.
$regSubKey.SetValue($regKey, $regValue,
[Microsoft.Win32.RegistryValueKind]::String)

# Close the key to release resources.
$regSubKey.Close()

Write-Host "Registry key created: $regPath\$regKey with value: $regValue"

pause # or Read-Host
```

## Delete Reg Key

---

```
# Compact Code
# Create a new PSDrive for HKEY_USERS.
if (-not (Get-PSDrive -Name HKU -PSProvider Registry -ErrorAction
'SilentlyContinue')) {
    New-PSDrive -Name HKU -PSProvider Registry -Root HKEY_USERS | Out-Null
}

$regPath = "SOFTWARE\Test"

$regHive =
[Microsoft.Win32.RegistryKey]::OpenBaseKey([Microsoft.Win32.RegistryHive]::CurrentUser, [Microsoft.Win32.RegistryView]::Registry64)

$regKey = $regHive.OpenSubKey("SOFTWARE", $true)
$regKey.DeleteSubKey("Test")

pause # or Read-Host
```

## Delete Reg Key Value

---

```
# Create a new PSDrive for HKEY_USERS.
if (-not (Get-PSDrive -Name HKU -PSProvider Registry -ErrorAction
'SilentlyContinue')) {
```



```
New-PSDrive -Name HKU -PSProvider Registry -Root HKEY_USERS | Out-Null}

# Define the Registry path and value details.
$regSubKey = "SOFTWARE\Test"
$regKey = "myKey"

$regHive =
[Microsoft.Win32.RegistryKey]::OpenBaseKey([Microsoft.Win32.RegistryHive]::CurrentUser, [Microsoft.Win32.RegistryView]::Registry64)

$regSubKey = $regHive.OpenSubKey("SOFTWARE\Test", $true)

if ($regSubKey) { $regSubKey.DeleteValue($regKey); $regSubKey.Close()
# $regHive.OpenSubKey("SOFTWARE", $true).DeleteSubKey("Test") # Delete the
entire 'Test' subkey
}
$regHive.Close()
pause # or Read-Host
```

Or

```
# Pop up, minus the specified 64 bit directive.
$regSubKey = "HKCU:\SOFTWARE\Test"

$regKey = Remove-ItemProperty $regSubKey -Name 'myKey'
$(New-Object -ComObject "WScript.Shell").Popup("The reg key was
deleted!", 0, "Reg Key", 0)

pause # or Read-Host
```

Or

```
# Define registry path and key name.
$regSubKey = "HKCU:\SOFTWARE\Test"
$regKey = "myKey"

try {
    # Ensure the Registry path exists before attempting to delete.
    if (Test-Path $regSubKey) {
        # Attempt to remove the Registry property.
        Remove-ItemProperty -Path "$regSubKey" -Name "$regKey" -ErrorAction
        'Stop'
        # Show the success popup.
        $(New-Object -ComObject "WScript.Shell").Popup("The registry key value
        was deleted!", 0, "Registry Key", 0)
    } else {
        # Show an error if the path does not exist.
        $(New-Object -ComObject "WScript.Shell").Popup("The registry path does
        not exist!", 0, "Error", 16)
    }
}
```



```
} catch {

    # Show error popup if something goes wrong.
    (New-Object -ComObject "WScript.Shell").Popup("Failed to delete the
    registry key value. Error: $_, 0, "Error", 16)

}

pause # or Read-Host
```

## Read Reg Key Value

---

```
# Compact Code
# Create a new PSDrive for HKEY_USERS.
if (-not (Get-PSDrive -Name HKU -PSProvider Registry -ErrorAction
    'SilentlyContinue')) {
    New-PSDrive -Name HKU -PSProvider Registry -Root HKEY_USERS | Out-Null
}

$regHive =
[Microsoft.Win32.RegistryKey]::OpenBaseKey([Microsoft.Win32.RegistryHive]::Curr
entUser, [Microsoft.Win32.RegistryView]::Registry64)

$regKey = $regHive.OpenSubKey("SOFTWARE\Test").GetValue("myKey", "Default
Value")

$regHive.Close()

Write-Host "myKey Value: $regKey"
pause # or Read-Host
```

Or

```
# Define the Registry path and key name.
$regSubKey = "HKCU:\SOFTWARE\Test"
$regKey = "myKey"

# Check if the Registry path exists and get the value.
if (Test-Path $regSubKey) {
    $regValue = (Get-ItemProperty -Path "$regSubKey" -Name $regKey).$regKey

    if ($Null -ne $regValue) {
        # Display the value of myKey in a popup.
        $(New-Object -ComObject "WScript.Shell").Popup("myKey: $regValue", 0,
        "Reg Key", 0)
    } else {
        # Display message if the key does not exist.
```



```
$ (New-Object -ComObject "WScript.Shell").Popup("myKey does not
exist.", 0, "Reg Key", 0)
}
} else {
    # Display message if the Registry path does not exist.
    $ (New-Object -ComObject "WScript.Shell").Popup("The registry path does not
exist.", 0, "Reg Key", 0)
}

pause # or Read-Host
```

Or

```
if (-not (Get-PSDrive -Name HKU -PSProvider Registry -ErrorAction
    'SilentlyContinue')) {
    New-PSDrive -Name HKU -PSProvider Registry -Root HKEY_USERS | Out-Null
}

# Open the Registry key in the 64-bit Registry view.
$regHive =
[Microsoft.Win32.RegistryKey]::OpenBaseKey([Microsoft.Win32.RegistryHive]::Curr
entUser, [Microsoft.Win32.RegistryView]::Registry64)

# Attempt to open the Registry subkey.
$regSubKey = $regHive.OpenSubKey("SOFTWARE\TEST")

if ($regSubKey -ne $Null) {
    # Try to get the value of "myKey".
    # Use $Null as default if the key does not exist.
    $regValue = $regSubKey.GetValue("myKey", $Null)

    if ($regValue -ne $Null) {
        Write-Host "The value of 'myKey' is: $regValue"
    } else {
        Write-Host "'myKey' does not exist under the specified registry path."
    }

    # Close the subkey to release resources.
    $regSubKey.Close()
} else {
    Write-Host "The registry path 'SOFTWARE\TEST' does not exist."
}

# Ensure the hive is properly closed to release resources.
$regHive.Close()

pause # or Read-Host
```



## Detect Reg Key, Return True or False

---

```
# Compact Code
# Create a new PSDrive for HKEY_USERS.
if (-not (Get-PSDrive -Name HKU -PSProvider Registry -ErrorAction
    'SilentlyContinue')) {
    New-PSDrive -Name HKU -PSProvider Registry -Root HKEY_USERS | Out-Null
}

$regHive =
[Microsoft.Win32.RegistryKey]::OpenBaseKey([Microsoft.Win32.RegistryHive]::Curr
entUser, [Microsoft.Win32.RegistryView]::Registry64)

$regSubKey = $regHive.OpenSubKey("SOFTWARE\Test")

$regKey = $regSubKey -and $regSubKey.GetValue("myKey", $Null) -ne $Null

$regHive.Close()

Write-Host "myKey exists: $regKey"
pause # or Read-Host
```

Or

```
$regSubKey = "HKCU:\SOFTWARE\Test"
$regKey = "myKey"

# Check if the registry key exists.
try {
    $regProp = (Get-ItemProperty -Path "$regSubKey" -ErrorAction
'Stop').PSObject.Properties.Name
    $regDetect = $regProp -contains $regKey
} catch {
    $regDetect = $false
}

$regDetect

pause # or Read-Host
```



## Detect Reg Key Data, Return True or False

---

```
# Compact Code
# Create a new PSDrive for HKEY_USERS if it doesn't exist.
if (-not (Get-PSDrive -Name HKU -PSProvider Registry -ErrorAction
    'SilentlyContinue')) {
    New-PSDrive -Name HKU -PSProvider Registry -Root HKEY_USERS | Out-Null
}

# Open the HKEY_CURRENT_USER hive in the 64-bit registry view.
$regHive =
[Microsoft.Win32.RegistryKey]::OpenBaseKey([Microsoft.Win32.RegistryHive]::CurrentUser, [Microsoft.Win32.RegistryView]::Registry64)

# Try to open the subkey.
$regSubKey = $regHive.OpenSubKey("SOFTWARE\Test")

# Check if the subkey and the value exist, and if the value equals 1.
$regKey = $regSubKey -and $regSubKey.GetValue("myKey", $Null) -eq 1

# Close the registry hive.
$regHive.Close()

# Output the result as True or False.
Write-Host "myKey's value is 1: $regKey"

pause # or Read-Host
```



## Add Reg Access Rule

---

```
# Create a new PSDrive for HKEY_USERS.
if (-not (Get-PSDrive -Name HKU -PSProvider Registry -ErrorAction
    'SilentlyContinue')) {
    New-PSDrive -Name HKU -PSProvider Registry -Root HKEY_USERS | Out-Null
}

$regHive =
[Microsoft.Win32.RegistryKey]::OpenBaseKey([Microsoft.Win32.RegistryHive]::Loca
lMachine, [Microsoft.Win32.RegistryView]::Registry64)

$regSubKey =
$regHive.OpenSubKey("SOFTWARE\Microsoft\Windows\CurrentVersion\Authentication\L
ogonUI")

$uSID = $regSubKey.GetValue("SelectedUserSID")

$regSubKey = "Registry::HKEY_USERS\$uSID\Software\Test"

$Acl = Get-Acl -Path "$regSubKey"

# Create the access rule to Deny 'SetValue' permission for the current user.
$denyRule = New-Object System.Security.AccessControl.RegistryAccessRule(
    "Users",
    [System.Security.AccessControl.RegistryRights]::FullControl,
    [System.Security.AccessControl.AccessControlType]::Deny
)

# Add the Deny rule to the ACL.
$Acl.AddAccessRule($denyRule)

# Apply the modified ACL back to the registry subkey.
Set-Acl -Path "$regSubKey" -AclObject $Acl

Write-Output "Deny rule for SetValue has been added successfully."

pause # or Read-Host
```



## Add Reg Access Rule, Everyone, Full Control

---

You may need to run this in the System Account, or check the execution policy.

```
# EVERYONE
$regHive =
[Microsoft.Win32.RegistryKey]::OpenBaseKey([Microsoft.Win32.RegistryHive]::LocalMachine, [Microsoft.Win32.RegistryView]::Registry64)

$regSubKey =
$regHive.OpenSubKey("SOFTWARE\Microsoft\Windows\CurrentVersion\Authentication\LogonUI")

$uSID = $regSubKey.GetValue("SelectedUserSID")

$regSubKey = "Registry::HKEY_USERS\$uSID\Software\Test"

# Define the registry path.
# Get the ACL of the registry key.
$Acl = Get-Acl -Path $regSubKey

$object = New-Object System.Security.Principal.NtAccount("EVERYONE")

# Create a new access rule granting full control.
$accessRule = New-Object System.Security.AccessControl.RegistryAccessRule(
    $object,
    "FullControl",
    [System.Security.AccessControl.InheritanceFlags]::ContainerInherit,
    [System.Security.AccessControl.PropagationFlags]::None,
    [System.Security.AccessControl.AccessControlType]::Allow
)

# Add the access rule to the ACL.
$Acl.SetAccessRule($accessRule)

# Apply the updated ACL to the registry key.
Set-Acl -Path $regSubKey -AclObject $Acl

# Display the updated ACL.
Get-Acl -Path $regSubKey | Format-List

pause # or Read-Host
```



## Remove Reg Access Rule

---

You may need to run this in the System Account, or check the execution policy. In my tests, if a Deny exists on yours or a built-in group you're a part of, the System Account will be required to bypass access restrictions.

```
# Command: PowerShell -ExecutionPolicy Bypass -File .\script.ps1
# ExecutionPolicy: Set-ExecutionPolicy -ExecutionPolicy Bypass -Scope
# Process -Force

# Test as System Account psexec64 -s cmd.exe

# Create a new PSDrive for HKEY_USERS.
if (-not (Get-PSDrive -Name HKU -PSProvider Registry -ErrorAction
    'SilentlyContinue')) {
    New-PSDrive -Name HKU -PSProvider Registry -Root HKEY_USERS | Out-Null
}

$regHive =
[Microsoft.Win32.RegistryKey]::OpenBaseKey([Microsoft.Win32.RegistryHive]::Loca
lMachine, [Microsoft.Win32.RegistryView]::Registry64)

$regSubKey =
$regHive.OpenSubKey("SOFTWARE\Microsoft\Windows\CurrentVersion\Authentication\L
ogonUI")

$uSID = $regSubKey.GetValue("SelectedUserSID")

$regSubKey = "Registry::HKEY_USERS\$uSID\Software\Test"

$Acl = Get-Acl -Path $regSubKey

# Scans for SetValue Deny.
$denyEntry = $Acl.Access | Where-Object { $_.AccessControlType -eq "Deny" }

# Removes SetValue Deny.
if ($denyEntry) {
    $Acl.RemoveAccessRule($denyEntry)
    Set-Acl -Path $regSubKey -AclObject $Acl
}
pause # or Read-Host
```

Or



```
# Uses System.Security.Principal.SecurityIdentifier("S-1-15-2-1").
# Create a new PSDrive for HKEY_USERS if it doesn't exist.
if (-not (Get-PSDrive -Name HKU -PSProvider Registry -ErrorAction
    'SilentlyContinue')) {
    New-PSDrive -Name HKU -PSProvider Registry -Root HKEY_USERS | Out-Null
}

Clear-Host

# Retrieve the user SID.
$regHive =
[Microsoft.Win32.RegistryKey]::OpenBaseKey([Microsoft.Win32.RegistryHive]::Loca
lMachine, [Microsoft.Win32.RegistryView]::Registry64)

$regSubKey =
$regHive.OpenSubKey("SOFTWARE\Microsoft\Windows\CurrentVersion\Authentication\L
ogonUI")

$uSID = $regSubKey.GetValue("SelectedUserSID")

# Define the registry key path.
$regSubKey = "HKU:\$uSID\Software\Test"

# Define the SID for 'ALL APPLICATION PACKAGES'.
$allAppPackagesSid = New-Object
System.Security.Principal.SecurityIdentifier("S-1-15-2-1")

# Get the ACL of the registry key.
$Acl = Get-Acl $regSubKey

# Find and remove the rule for 'ALL APPLICATION PACKAGES' by SID.
$Rules = $Acl.Access | Where-Object { $_.IdentityReference.Value -Match "ALL
APPLICATION PACKAGES" }

ForEach ($Rule in $Rules) {
    Write-Host "Found rule: $($Rule.IdentityReference)"

    # Create the exact access rule using the SID.
    $ruleToRemove = New-Object
    System.Security.AccessControl.RegistryAccessRule(
        $allAppPackagesSid,
        $Rule.RegistryRights,
        $Rule.AccessControlType
    )

    # Remove the rule from the ACL.
}
```



```
$Acl.RemoveAccessRule($ruleToRemove)

Write-Host "Removed access for 'APPLICATION PACKAGE AUTHORITY\ALL
APPLICATION PACKAGES'"

}

# Apply the updated ACL to the registry key.
Set-Acl -Path $regSubKey -AclObject $Acl

# Check the updated ACL.
$updatedAcl = Get-Acl $regSubKey

Write-Host "Updated ACL: "

$updatedAcl.Access | ForEach-Object {
    Write-Host "$($_.IdentityReference) - $($_.RegistryRights)"
}

pause # or Read-Host
```



## Rebuild Reg SubKey ACLs

---

```
# Create a new PSDrive for HKEY_USERS if it does not exist.
if (-not (Get-PSDrive -Name HKU -PSProvider Registry -ErrorAction
    'SilentlyContinue')) {
    New-PSDrive -Name HKU -PSProvider Registry -Root HKEY_USERS | Out-Null
}

Clear-Host

# Retrieve the user SID.
$regHive =
[Microsoft.Win32.RegistryKey]::OpenBaseKey([Microsoft.Win32.RegistryHive]::Loca
lMachine, [Microsoft.Win32.RegistryView]::Registry64)

$regSubKey =
$regHive.OpenSubKey("SOFTWARE\Microsoft\Windows\CurrentVersion\Authentication\L
ogonUI")

$uSID = $regSubKey.GetValue("SelectedUserSID")

# Define the registry key path.
$regSubKey = "HKU:\$uSID\Software\Test"

# Define the SID for 'ALL APPLICATION PACKAGES'.
$allAppPackagesSid = New-Object
System.Security.Principal.SecurityIdentifier("S-1-15-2-1")

# Get the ACL of the registry key
$Acl = Get-Acl $regSubKey

# Force removal of the rule by resetting the ACL.
Write-Host "Resetting ACL to remove all rules, including for ALL APPLICATION
PACKAGES."

# First, clear all existing rules from the ACL.
# Disable inheritance and remove inherited rules.
$Acl.SetAccessRuleProtection($true, $false)

# Now remove the rule for 'ALL APPLICATION PACKAGES'
$Acl.Access | Where-Object { $_.IdentityReference.Value -eq
$allAppPackagesSid.Value } | ForEach-Object {
    Write-Host "Removing rule for SID: $($allAppPackagesSid.Value)"
    $Acl.RemoveAccessRule($_)
}
```



```
# Apply the modified ACL to the registry key.
Set-Acl -Path $regSubKey -AclObject $Acl

# Now check and display the final ACL after the reset.
$finalAcl = Get-Acl $regSubKey

Write-Host "Final ACL after resetting and removing the principal:"
$finalAcl.Access | ForEach-Object {
    Write-Host "$($_.IdentityReference) - $($_.RegistryRights)"
}

# Reapply a clean ACL without the removed rule, ensuring no other
# unexpected rules remain.
$cleanAcl = New-Object System.Security.AccessControl.RegistrySecurity

# Add access rules with the correct syntax.
$cleanAcl.AddAccessRule([System.Security.AccessControl.RegistryAccessRule]::New
("Everyone", "ReadKey", "Allow"))

# Apply the updated ACL to the registry key.
Set-Acl -Path $regSubKey -AclObject $cleanAcl

# Check the final clean ACL.
$finalCleanAcl = Get-Acl $regSubKey

Write-Host "Final Clean ACL:"
$finalCleanAcl.Access | ForEach-Object {
    Write-Host "$($_.IdentityReference) - $($_.RegistryRights)"
}

pause # or Read-Host
```



## Return Reg AccessControlType

```
# Create a new PSDrive for HKEY_USERS.
if (-not (Get-PSDrive -Name HKU -PSProvider Registry -ErrorAction
    'SilentlyContinue')) {
    New-PSDrive -Name HKU -PSProvider Registry -Root HKEY_USERS | Out-Null
}

# Open the Local Machine registry hive in 64-bit view.
$regHive =
[Microsoft.Win32.RegistryKey]::OpenBaseKey([Microsoft.Win32.RegistryHive]::Loca
lMachine, [Microsoft.Win32.RegistryView]::Registry64)

# Access the LogonUI registry key to retrieve user session data.
$regSubKey =
$regHive.OpenSubKey("SOFTWARE\Microsoft\Windows\CurrentVersion\Authentication\LogonUI")

# Return the user SID.
$uSID = $regSubKey.GetValue("SelectedUserSID")

# Construct the HKEY_USERS with user SID.
$regSubKey = "Registry::HKEY_USERS\$uSID\Software\Test"

# Get the ACL for the registry key.
$Acl = Get-Acl -Path $regSubKey

# Filter and display user ACLs.
$Acl.Access | Where-Object { $_.AccessControlType -eq "Deny" } |
Foreach-Object {
    Write-Host "Identity Reference: $($_.IdentityReference)"
    Write-Host "Access Control Type: $($_.AccessControlType)"
    Write-Host "===="
}
}

pause # or Read-Host
```



## Change Reg Key Owner

### ESCAPE CHARACTERS

```
# DOMAIN USER
$Acl = Get-Acl "Registry::HKCU\Software\Test"

$object = New-Object System.Security.Principal.NtAccount("Domain-Here",
"Username-Here")
# Example: "BUILTIN", "Administrators"

$Acl.SetOwner($object)

# Apply the updated ACL to the registry key.
$Acl | Set-Acl "Registry::HKCU\Software\Test"

$Acl = Get-Acl "Registry::HKCU\Software\Test" | Format-List
$Acl; pause # or Read-Host
```

## Disable Reg Key Inheritance

```
# Create a new PSDrive for HKEY_USERS.
if (-not (Get-PSDrive -Name HKU -PSProvider Registry -ErrorAction
'SilentlyContinue')) {
    New-PSDrive -Name HKU -PSProvider Registry -Root HKEY_USERS | Out-Null
}

# Open the Local Machine registry hive in 64-bit view.
$regHive =
[Microsoft.Win32.RegistryKey]::OpenBaseKey([Microsoft.Win32.RegistryHive]::Loca
lMachine, [Microsoft.Win32.RegistryView]::Registry64)

# Access the LogonUI registry key to retrieve user session data.
$regSubKey =
$regHive.OpenSubKey("SOFTWARE\Microsoft\Windows\CurrentVersion\Authentication\L
ogonUI")

# Return the user SID.
$uSID = $regSubKey.GetValue("SelectedUserSID")

# Construct the HKEY_USERS with user SID.
$regSubKey = "Registry::HKEY_USERS\$uSID\Software\Test"

# Disable Inheritance
$Acl = Get-Acl -Path $regSubKey
```



```
$Acl.SetAccessRuleProtection($true, $false)

# Apply the updated ACL to the registry key.
$Acl | Set-Acl $regSubKey

pause # or Read-Host
```

## Delete Reg Key User Principal

---

```
# Delete NT AUTHORITY\System
# Create a new PSDrive for HKEY_USERS.
if (-not (Get-PSDrive -Name HKU -PSProvider Registry -ErrorAction
    'SilentlyContinue')) {
    New-PSDrive -Name HKU -PSProvider Registry -Root HKEY_USERS | Out-Null
}

$regHive =
[Microsoft.Win32.RegistryKey]::OpenBaseKey([Microsoft.Win32.RegistryHive]::Loca
lMachine, [Microsoft.Win32.RegistryView]::Registry64)

$regSubKey =
$regHive.OpenSubKey("SOFTWARE\Microsoft\Windows\CurrentVersion\Authentication\L
ogonUI")

# Return the user SID.
$uSID = $regSubKey.GetValue("SelectedUserSID")

$regSubKey = "Registry::HKEY_USERS\$uSID\Software\Test"

# Delete specific user principal.
$Acl = Get-Acl -Path $regSubKey

# Retrieve the existing ACL rule for the "NT AUTHORITY\System" account.
$Rule = $Acl.Access | Where-Object { $_.IdentityReference -eq "NT
AUTHORITY\System" }

# Check if the rule exists before attempting to remove it.
if ($Rule) {
    $Acl.RemoveAccessRule($Rule)
}

# Apply the updated ACL to the registry key.
$Acl | Set-Acl $regSubKey

pause # or Read-Host
```



## Return Reg SubKey Principals

---

```
Clear-Host

# Create a new PSDrive for HKEY_USERS if it does not exist.
if (-not (Get-PSDrive -Name HKU -PSProvider Registry -ErrorAction
    'SilentlyContinue')) {
    New-PSDrive -Name HKU -PSProvider Registry -Root HKEY_USERS | Out-Null
}

# Retrieve the user SID.
$regHive =
[Microsoft.Win32.RegistryKey]::OpenBaseKey([Microsoft.Win32.RegistryHive]::Loca
lMachine, [Microsoft.Win32.RegistryView]::Registry64)

$regSubKey =
$regHive.OpenSubKey("SOFTWARE\Microsoft\Windows\CurrentVersion\Authentication\L
ogonUI")

# Return the user SID.
$uSID = $regSubKey.GetValue("SelectedUserSID")

# Construct the HKEY_USERS with user SID.
$regSubKeyPath = "HKU:\$uSID\Software\Test"

$Acl = Get-Acl -Path $regSubKeyPath

# Loop through the access rules and output only the IdentityReference.
$Acl.Access | ForEach-Object { ($_.IdentityReference).value }

pause # or Read-Host
```

## Return Reg SubKey Principals Information

---

```
# Ensure PSDrive for HKEY_USERS exists.
if (-not (Get-PSDrive -Name HKU -PSProvider Registry -ErrorAction
    'SilentlyContinue')) {
    New-PSDrive -Name HKU -PSProvider Registry -Root HKEY_USERS | Out-Null
}

# Retrieve the User SID.
$regHive =
[Microsoft.Win32.RegistryKey]::OpenBaseKey([Microsoft.Win32.RegistryHive]::Loca
lMachine, [Microsoft.Win32.RegistryView]::Registry64)
```



```
$regSubKey =
$regHive.OpenSubKey("SOFTWARE\Microsoft\Windows\CurrentVersion\Authentication\LogonUI")

# Return the user SID.
$uSID = $regSubKey.GetValue("SelectedUserSID")

$regHive.Close()

# Construct the HKEY_USERS with user SID.
$regSubKeyPath = "HKU:\$uSID\Software\Test"

try {

# Retrieve the Access Control List (ACL) of the registry key.
$Acl = Get-Acl -Path $regSubKeyPath

# Extract and display the principals (IdentityReference).
Write-Output "Principals for '$regSubKeyPath':"
$Acl.Access | ForEach-Object {
    [PSCustomObject]@{
        Principal = $_.IdentityReference
        AccessType = $_.AccessControlType
        Rights     = $_.RegistryRights
    }
} | Format-Table -AutoSize
}

catch {
    Write-Error "Failed to retrieve principals for '$regSubKeyPath': $_"
}

pause # or Read-Host
```

## Add Reg Principal and ACE Rule

---

```
Clear-Host

# Create a new PSDrive for HKEY_USERS.
if (-not (Get-PSDrive -Name HKU -PSProvider Registry -ErrorAction
'SilentlyContinue')) {
    New-PSDrive -Name HKU -PSProvider Registry -Root HKEY_USERS | Out-Null
}

# Retrieve the user SID.
```



```
$regHive =
[Microsoft.Win32.RegistryKey]::OpenBaseKey([Microsoft.Win32.RegistryHive]::LocalMachine, [Microsoft.Win32.RegistryView]::Registry64)

$regSubKey =
$regHive.OpenSubKey("SOFTWARE\Microsoft\Windows\CurrentVersion\Authentication\LogonUI")

# Return the user SID.
$uSID = $regSubKey.GetValue("SelectedUserSID")
if (-not $uSID) {
    Write-Host "Error: Unable to retrieve the user SID."
    return
}

# Retrieve the logged-on user name.
$SessionPath =
"HKLM:\SOFTWARE\Microsoft\Windows\CurrentVersion\Authentication\LogonUI\SessionData\1"

$Username = (Get-ItemProperty -Path $SessionPath -Name 'LoggedOnSAMUser' -ErrorAction Stop).LoggedOnSAMUser

# Define the registry key path.
$regSubKeyPath = "HKU:\$uSID\Software\Test"

# Ensure the registry key exists.
if (-not (Test-Path -Path $regSubKeyPath)) {
    Write-Host "Registry path '$regSubKeyPath' does not exist."
    return
}

# Get the ACL for the registry key.
$Acl = Get-Acl -Path $regSubKeyPath

# Disable Inheritance
$Acl.SetAccessRuleProtection($true, $false)

# Add FullControl for the current user.
$AddACL1 = New-Object
System.Security.AccessControl.RegistryAccessRule("$Username", "FullControl", "Allow")

$Acl.AddAccessRule($AddACL1)

# Add FullControl for local administrators.
```



```
$AddACL2 = New-Object  
System.Security.AccessControl.RegistryAccessRule("BUILTIN\Administrators",  
"FullControl", "Allow")  
  
$Acl.AddAccessRule($AddACL2)  
  
# Apply the updated ACL to the registry key.  
Set-Acl -Path $regSubKeyPath -AclObject $Acl  
  
# Display the updated ACL.  
Get-Acl -Path $regSubKeyPath | Format-List  
  
Write-Host "ACL updated successfully for $regSubKeyPath."  
pause # or Read-Host
```

## NOTE

---

If you're looking for explicit control or ensuring you follow a more detailed approach (especially when dealing with complex access control or account handling), it's better to use the *NTAccount* constructor.

```
$AddACL2 = New-Object System.Security.AccessControl.RegistryAccessRule(  
    (New-Object System.Security.Principal.NTAccount("BUILTIN",  
    "Administrators")),  
    "FullControl",  
    "Allow"  
)
```



## Update or Remove Reg ACE Rule

```
Clear-Host
```

```
# Create a new PSDrive for HKEY_USERS if it does not exist.
if (-not (Get-PSDrive -Name HKU -PSProvider Registry -ErrorAction
    'SilentlyContinue')) {
    New-PSDrive -Name HKU -PSProvider Registry -Root HKEY_USERS | Out-Null
}

# Retrieve the user SID.
$regHive =
[Microsoft.Win32.RegistryKey]::OpenBaseKey([Microsoft.Win32.RegistryHive]::Loca
lMachine, [Microsoft.Win32.RegistryView]::Registry64)

$regSubKey1 =
$regHive.OpenSubKey("SOFTWARE\Microsoft\Windows\CurrentVersion\Authentication\L
ogonUI")

$uSID = $regSubKey1.GetValue("SelectedUserSID")

if (-not $uSID) {
    Write-Host "Error: Unable to retrieve the user SID."
    return
}

# Retrieve the logged-on user name.
# Define the registry key path.
$regSubKey2 = "HKU:\$uSID\Software\Test"

# Ensure the registry key exists.
if (-not (Test-Path -Path $regSubKey2)) {
    Write-Host "Registry path '$regSubKey2' does not exist."
    return
}

# Get the ACL for the registry key.
$Acl = Get-Acl -Path $regSubKey2

# Remove "FullControl" for the "Everyone" principal.
$Rules = $Acl.Access | Where-Object { $_.IdentityReference -eq "Everyone" }

ForEach ($Rule in $Rules) {
    # Remove the existing ACE for "Everyone"
    $Acl.RemoveAccessRule($Rule)
```



}

```
# Add a new ACE for "Everyone" with ".ReadKey" permission.  
$newRule = New-Object System.Security.AccessControl.RegistryAccessRule(  
    "Everyone",  
    ".ReadKey",  
    "Allow"  
)  
  
$Acl.AddAccessRule($newRule)  
  
# Apply the updated ACL to the registry key.  
Set-Acl -Path $regSubKey2 -AclObject $Acl  
  
# Display the updated ACL.  
Write-Host "Updated ACL for $($regSubKeyPath)"  
  
Get-Acl -Path $regSubKey2 | Format-List  
  
Write-Host "ACL updated successfully for $regSubKey2."  
pause # or Read-Host
```

## Add Reg Key for All Users

---

```
Clear-Host  
  
# Enable strict error handling.  
$ErrorActionPreference = 'Stop'  
  
# Retrieve the User SID from the registry.  
$LogonUIPath =  
    "HKLM:\Software\Microsoft\Windows\CurrentVersion\Authentication\LogonUI"  
$UserSID = (Get-ItemProperty -Path $LogonUIPath -Name 'SelectedUserSID'  
    -ErrorAction 'SilentlyContinue').SelectedUserSID  
  
if ($UserSID) {  
    Write-Host "User SID: $UserSID"  
  
    # Define the target path in HKEY_USERS.  
    $RegistryPath = "Registry::HKEY_USERS\$UserSID\Software\TEST"  
  
    # Create the registry key if it doesn't exist.  
    if (-not (Test-Path $RegistryPath)) {
```



```
Write-Host "Creating registry path: $RegistryPath"

New-Item -Path $RegistryPath -Force | Out-Null
}

# Set the registry value.
Set-ItemProperty -Path $RegistryPath -Name "myKey" -Value 1 -Force
Write-Host "Registry key added successfully: $RegistryPath"
} else {
    Write-Warning "User SID not found. Ensure the SelectedUserSID exists in the
LogonUI registry key."
}

Write-Host "`nDone!"
pause # or Read-Host
```

## Delete Acrobat Reg Keys, Recursively

---

This is a practical example of a task I had to address. During testing of an Acrobat App Attach installation, all the Start Menu icons were orphaned. This script was instrumental in removing the leftover registry keys that were causing the issue.

```
Clear-Host

# Enable strict error handling.
$ErrorActionPreference = 'Stop'

# Define the registry root path and the subkey name to search and delete.
$registryPath      = "HKLM:\SOFTWARE\Classes"
$subkeyToDelete   = "AdobeAcrobat64Bit_24.5.20320.0_x64__lnrb0vh0kd882"

# Function to recursively scan and force-remove subkeys based on name match.
function Remove-RegistrySubkeys {
    param (
        [string]$path,
        [string]$subkeyName
    )

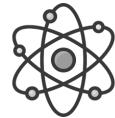
    # Get subkeys at the current path (recursively), filter by subkey name.
    $subkeys = Get-ChildItem -Path $path -Recurse -ErrorAction
    'SilentlyContinue' | Where-Object { $_.PSChildName -eq $subkeyName }
```



```
foreach ($subkey in $subkeys) {
    try {
        Write-Host "Force removing registry subkey: $($subkey.PSPPath)"
        Remove-Item -Path $subkey.PSPPath -Recurse -Force -ErrorAction
        'Stop'
    }
    catch {
        Write-Host "Failed to remove registry subkey: $($subkey.PSPPath) - "
        $_
    }
}

# Start the recursive force removal process from the root registry path.
Remove-RegistrySubkeys -Path $registryPath -SubkeyName $subkeyToDelete

pause # or Read-Host
```



## PowerShell: Logical Constructs

---

How to test from the System Account ([PsExec](#)):

```
PsExec64 -S CMD  
"C:\PowerShell\script.ps1"
```

### IF Condition

---

The **IF Condition** is a fundamental control flow construct in programming that executes a block of code only when a specified condition is met. It enables programs to make decisions dynamically based on given conditions. In PowerShell, the **IF** statement can be combined with cmdlets to create powerful and flexible expressions.

```
$procName = "notepad.exe"

# Check if the process is running.
if (Get-Process -Name ($procName -Replace '\.exe$', '') -ErrorAction
'SilentlyContinue') {

    Write-Output "$procName is running.

}

pause
```

### TEMPLATE

---

```
if ( condition ) { Your Code Here }
```



## IF-ELSE Condition

---

The **IF-ELSE Condition** is a control flow statement in programming that allows the execution of a block of code based on whether a specified condition is true or false. If the condition evaluates to true, one block of code is executed; otherwise, an alternative block of code is executed. This structure enables decision-making in programs by defining different actions depending on the outcome of the condition.

Clear-Host

```
$number = Read-Host "Enter a number`n"

if ($number -gt 10) {

    Write-Output "The number is greater than 10."

} else {

    Write-Output "The number is 10 or less."
}

pause
```

## TEMPLATE

---

```
if ( condition ) { Your Code Here } else { Your Code Here }
```



## IF-ELSEIF Condition

---

The **IF-ELSEIF Condition** is a control flow statement that enables programs to evaluate multiple conditions sequentially. If the first condition is true, its corresponding block of code executes. If not, the program checks the next condition, continuing until a match is found or the final `else` block runs. This structure allows for more complex decision-making by handling multiple possible outcomes efficiently.

```
Clear-Host

$score = Read-Host "Enter your score`n"

if ($score -ge 90) {
    Write-Output "Grade: A"
}

elseif ($score -ge 80) {
    Write-Output "Grade: B"
}

elseif ($score -ge 70) {
    Write-Output "Grade: C"
}

elseif ($score -ge 60) {
    Write-Output "Grade: D"
}

else {
    Write-Output "Grade: F"
}

pause
```

## TEMPLATE

---

```
if ( condition 1 ) { Your Code Here 1 }
elseif ( condition 2 ) { Your Code Here 2 }
elseif ( condition 3 ) { Your Code Here 3 }
```



## SWITCH Statement, Decision-making

---

The **SWITCH Statement** in PowerShell is a control flow construct that will help simplify decision-making when evaluating multiple possible values. Unlike if-elseif, which checks conditions sequentially, switch efficiently compares an expression against multiple cases and executes the matching block of code. This makes it useful for handling scenarios with distinct, predefined options in a more readable and concise manner.

```
Clear-Host
```

```
$menuOption = Read-Host "Enter a number (1-7) to represent a menu option`n"

switch ($menuOption) {
    1 { Write-Output "Option 1"; break }
    2 { Write-Output "Option 2"; break }
    3 { Write-Output "Option 3"; break }
    4 { Write-Output "Option 4"; break }
    5 { Write-Output "Option 5"; break }
    6 { Write-Output "Option 6"; break }
    7 { Write-Output "Option 7"; break }

    default {
        Write-Output "Invalid input. Enter a number between 1 and 7."
    }
}

pause
```

---

## TEMPLATE

---

```
switch ( test statement ) {
    match 1 { statement 1; Your Code Here }
    match 2 { statement 2; Your Code Here }
    default { statement 0; Your Code Here }
}
```



## FOR Statement, Loop

---

The **FOR Statement** in PowerShell is a looping construct that allows code to be executed repeatedly based on a defined condition. It consists of an initialization, a condition check, and an iteration step, making it ideal for scenarios where the number of iterations is known in advance. This structure provides efficient control over loop execution, enabling automation and repetitive task handling in scripts.

```
Clear-Host

$servers = @("Server01", "Server02", "Server03", "Server04")

# Loop through each server in the list.
for ($i = 0; $i -lt $servers.Length; $i++) {

    $server = $servers[$i]

    # Test the connection to the server using Ping.
    if (Test-Connection -ComputerName $server -Count 1 -Quiet) {

        Write-Output "$server is ONLINE"

    } else {

        Write-Output "$server is OFFLINE"

    }
    Start-Sleep 2
}

pause
```

## TEMPLATE

---

```
for (initialize; condition; repeat) { Your Code Here }

$item = @("A","B","C")
for ($i = 0; $i -lt $item.Count; $i++) { $item[$i] }
```



## FOREACH Statement, Loop

---

The **ForEach Statement** in PowerShell is a looping construct designed to iterate over each element in a collection, such as an array, list, or object. Unlike the for statement, which requires explicit control over initialization and iteration, **ForEach** simplifies iteration by automatically processing each item in the collection. It is especially useful for tasks that involve applying an action to every element in a set, making it an efficient and concise choice for processing items in scripts.

```
Clear-Host

# Define an array of names.
$names = @("Alice", "Bob", "Charlie", "David", "Eve")

# Use ForEach to iterate through each name and display a greeting.
ForEach ($name in $names) {

    Write-Output "Hello, $name!`n"

    Start-Sleep 2

    # Your code here

}

pause
```

## TEMPLATE

---

```
ForEach (item in collection) { Your Code Here }
```



## WHILE Statement, Loop

---

The **While Statement** in PowerShell is a looping construct that repeatedly executes a block of code as long as a specified condition evaluates to true. The loop continues to run until the condition becomes false, making it useful for situations where the number of iterations is not known in advance but depends on dynamic conditions. This structure allows for flexible control over loop execution, enabling actions to be performed continuously while the condition holds true.

```
Clear-Host

# Create a counter, while.
# Initialize the counter.
$counter = 1

# While loop that runs as long as the counter is less than or equal to 5.
while ($counter -le 10) {

    Write-Output "Counter is at $counter"

    # Your Code Here

    Start-Sleep 2

    $counter++ # Increment the counter.

}

pause
```

## TEMPLATE

---

```
while ( condition ) { Your Code Here }
```



## TRY-CATCH Block, Exception Handling

---

A **Try-Catch Block** is used for exception handling. It allows you to run code in a "try" block and catch any errors that occur during execution in the corresponding "catch" block. If an error occurs in the try block, the script will jump to the catch block, where you can handle the error, log it, or display a custom message. Okay, while not strictly a logical construct, a **Try-Catch Block** can be considered a logical construct in the sense that it controls the flow of your program based on whether an error occurs.

```
Clear-Host
```

```
try {
    # Try to open a file that may not exist.
    $content = Get-Content "nonexistentfile.txt"
}
catch {
    # If an error occurs (like file not found), handle it here.
    Write-Host "The file does not exist."
}

pause
```

## TEMPLATE

---

```
try {
    # Your Code Here
}

catch {
    # Your Code Here
}
```



## PowerShell: Data Structures

---

### Arrays

---

An **Array** in PowerShell is a flexible atomic data structure that stores an ordered collection of elements. Unlike Batch scripting, where handling multiple values often relies on parsing strings or using separate variables, PowerShell provides native support for arrays. Unlike traditional arrays in lower-level languages, PowerShell arrays can hold elements of different types, grow dynamically when needed, and offer powerful built-in methods for manipulation. Array indexing starts at 0, and elements can be accessed, modified, and iterated efficiently using PowerShell's object-oriented features.

```
Clear-Host
```

```
try {
    # Define an array with some elements.
    $numbers = @(5, 10, 15, 20)

    # Prompt user for an index.
    $index = Read-Host "Enter an index to retrieve"

    # Access the array element.
    if ($index -ge 0 -and $index -lt $numbers.Count) {
        Write-Host "Value at index $index is: $($numbers[$index])"
    } else {
        throw "Index out of range."
    }
}
catch {
    # Our error handling.
    Write-Host "Error: $_"
}

pause
```



## TEMPLATE

---

### Basic Syntax

```
# Define an array.  
$array = @("Item1", "Item2", "Item3")  
  
# Perform an operation.  
$result = $array[5]  
  
# Display the result.  
Write-Host "Operation successful: $result"
```

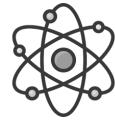
### Read Text File

```
# Read lines from a text file into an array.  
$array = Get-Content "data.txt"  
  
# Perform an operation.  
$result = $array[5]  
  
# Display the result.  
Write-Host "Operation successful: $result"
```

### NOTE

---

[https://learn.microsoft.com/en-us/powershell/module/microsoft.powershell.core/about/about\\_arrays?view=powershell-7.5](https://learn.microsoft.com/en-us/powershell/module/microsoft.powershell.core/about/about_arrays?view=powershell-7.5)



## Stacks

---

A **Stack** in PowerShell follows the Last In, First Out (LIFO) principle. Unlike Batch, where simulating a stack requires manual management of variables and counters, PowerShell provides an easier way to implement a stack using arrays or the [System.Collections.Stack] class. You can push elements onto the stack using .Push() and remove them using .Pop(), making it efficient for managing recursive operations or undo functionality.

```
Clear-Host

try {
    # Create a stack.
    $stack = New-Object System.Collections.Stack

    # Simulate user actions.
    $stack.Push("Opened File")
    $stack.Push("Edited Line 1")
    $stack.Push("Deleted Paragraph")
    $stack.Push("Saved File")

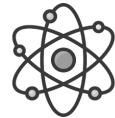
    Write-Host "Current Actions: $($stack -Join ', ')"

    # Perform an undo operation (removing the last action).
    $lastAction = $stack.Pop()
    Write-Host "`nUndoing Last Action: $lastAction"

    Write-Host "Remaining Actions: $($stack -Join ', ')"
}

catch {
    # Handle any errors.
    Write-Host "An error occurred: $_"
}

pause
```



## TEMPLATE

---

```
# Create Stack
$stack = New-Object System.Collections.Stack

# Push elements onto the stack.
$stack.Push("Element1")
$stack.Push("Element2")
$stack.Push("Element3")

# Pop an element from the stack
$popped = $stack.Pop()

# Display Result
Write-Host "Popped item: $popped"
```

## NOTE

---

<https://learn.microsoft.com/en-us/powershell/module/microsoft.powershell.management/push-location?view=powershell-7.5>

<https://learn.microsoft.com/en-us/powershell/module/microsoft.powershell.management/pop-location?view=powershell-7.5>



## Queues

---

A **Queue** in PowerShell follows the First In, First Out (FIFO) principle. While Batch requires manual index tracking, PowerShell offers a built-in [System.Collections.Queue] class for easy queue management. You can enqueue elements using `.Enqueue()` and dequeue them using `.Dequeue()`, allowing for efficient task scheduling, message processing, and other sequential operations.

```
Clear-Host

try {
    # Create a queue.
    $queue = New-Object System.Collections.Queue

    # Simulate adding the print jobs.
    $queue.Enqueue("Document1.pdf")
    $queue.Enqueue("Report.docx")
    $queue.Enqueue("Presentation.pptx")

    Write-Host "Print Queue: $($queue -Join ', ')"

    # Process the print jobs (FIFO).
    while ($queue.Count -gt 0) {
        $job = $queue.Dequeue()
        Write-Host "`nProcessing: $job"
        Start-Sleep -Seconds 1 # Simulating print time.
    }

    Write-Host "`nAll print jobs processed."
}
catch {
    # Handle Errors
    Write-Host "An error occurred: $_"
}

pause
```



## TEMPLATE

---

```
# Create Queue
$queue = New-Object System.Collections.Queue

# Enqueue Elements
$queue.Enqueue("Item1")
$queue.Enqueue("Item2")
$queue.Enqueue("Item3")

# Dequeue Element (FIFO)
$dequeued = $queue.Dequeue()

# Display Result
Write-Host "Dequeued item: $dequeued"
```



## Hash Tables

---

A **Hash Table** in PowerShell is an efficient data structure for storing key-value pairs. Unlike Batch, where key-value storage is limited and requires manual parsing, PowerShell provides a built-in [Hashtable] type. Hash Tables allow fast lookups, insertions, and deletions using keys. They are particularly useful for configuration settings, caching, and data lookups, often operating in constant time. You can define a hash table using @{ Key = Value } syntax and access values using \$hashTable["Key"].

```
Clear-Host
```

```
try {
    # Create a hash table to store user information.
    $users = @{}

    # Add users (Key: Username, Value: Email).
    $users["jdoe"] = "jdoe@domain.com"
    $users["bsmith"] = "bsmith@domain.com"
    $users["kjones"] = "kjones@domain.com"

    Write-Host "User Directory:`n"
    $users.GetEnumerator() | ForEach-Object { Write-Host "$($_.Key) - 
    $($_.Value)" }

    # Retrieve a user's email.
    $username = "jdoe"
    if ($users.ContainsKey($username)) {
        Write-Host "`nEmail for $username: $($users[$username])"
    } else {
        Write-Host "`nUser not found."
    }
}
catch {
    # Handle Errors
    Write-Host "An error occurred: $_"
}

pause
```



## TEMPLATE

---

```
# Create Hash Table
$hashTable = @{}

# Add the key-value pairs.
$hashTable["Key1"] = "Value1"
$hashTable["Key2"] = "Value2"
$hashTable["Key3"] = "Value3"

# Retrieve a value by key.
$value = $hashTable["Key2"]
Write-Host "Retrieved Value: $value"

# Remove a key-value pair.
$hashTable.Remove("Key1")

# Display the remaining hash table contents.
Write-Host "`nUpdated Hash Table:"
$hashTable.GetEnumerator() | ForEach-Object { Write-Host "$($_.Key) - 
$($_.Value)" }
```



## PowerShell: Miscellaneous

---

How to test from the System Account (**PsExec**):

```
PsExec64 -S CMD  
PowerShell -ExecutionPolicy Bypass -File "C:\PowerShell\script.ps1"
```

### Create a Counter, For

---

```
# Initialize Counter  
$Counter = 0  
  
# Start a loop that runs 10 times.  
for ($i = 1; $i -le 10; $i++) {  
    # Increment the counter.  
    $Counter++  
  
    # Display the current value of the counter.  
    Write-Host "Counter is now: $Counter"  
}  
  
Write-Host "Final counter value: $Counter"  
pause # or Read-Host
```

### Detect if 32 Bit or 64 Bit Environment

---

Checks the system's architecture and displays '32 Bit' if it's 32-bit or '64 Bit' if it's 64-bit.

```
if ([IntPtr]::Size -eq 4) { Write-Host "32 Bit" } else { Write-Host "64 Bit" }  
  
pause # or Read-Host
```

### Create Event Log

---

```
New-EventLog -LogName 'Application' -Source 'AppName1' -ErrorAction  
'SilentlyContinue'
```



```
Write-Eventlog -LogName 'Application' -Message 'Your Message Goes Here' -Source  
'AppName1' -id 777 -EntryType 'Information' -Category 0  
  
pause # or Read-Host
```

## Random Beeps

---

```
100..2000 | Get-Random -Count 30 | ForEach {  
    [console]::Beep($_, 100)  
}  
pause # or Read-Host
```

## Add Sound

---

```
[System.Media.SystemSounds]::Asterisk.Play()  
Start-Sleep 2  
[System.Media.SystemSounds]::Beep.Play()  
Start-Sleep 2  
[System.Media.SystemSounds]::Exclamation.Play()  
Start-Sleep 2  
[System.Media.SystemSounds]::Hand.Play()  
Start-Sleep 2  
[System.Media.SystemSounds]::Question.Play()  
  
$playWav = New-Object System.Media.SoundPlayer  
$playWav.SoundLocation = 'C:\Windows\Media\dm\Windows Background.wav'  
$playWav.PlaySync()  
  
Add-Type -AssemblyName System.Speech  
$synth = New-Object -TypeName System.Speech.Synthesis.SpeechSynthesizer  
$synth.GetInstalledVoices() | Select-Object -ExpandProperty 'VoiceInfo'  
$synth.SelectVoice("Microsoft Zira Desktop")  
$synth.Rate = 1  
$synth.Volume = 100  
  
$synth.Speak("Hello User")  
$synth.Speak("The script has finished!")  
  
pause # or Read-Host
```



## Using -NotLike to Filter Results

---

```
$groupList = @("Group1", "Group2", "Group3")

# Check if none of the items contain "DESKTOP".
if ($groupList -NotLike "*DESKTOP*") {
    Write-Output "No group contains 'DESKTOP'"
} else {
    Write-Output "A group contains 'DESKTOP'"
}
pause # or Read-Host
```

Or

```
# Return Group Members
$adminGroupCmd = & C:\Windows\system32\NET LOCALGROUP Administrators

# Split
$Lines = $adminGroupCmd -Split "`n"

# Find "Members"
$startIndex = [Array]::IndexOf($Lines, "Members")

# Extract usernames, excluding unwanted lines.
$Usernames = $Lines[($startIndex + 1)..($Lines.Length - 1)] | Where-Object {
$_ -Match '^[\s]*\w' `-
-and $_ -NotLike "*The command completed successfully*" `-
-and $_ -NotLike "*DOMAIN*" `-
-and $_ -NotLike "*DESKTOP*" `-
-and $_ -NotLike "*Administrator*" } | ForEach-Object { $_.Trim()
}

pause # or Read-Host
```

## Return Chrome Version

---

```
# Return Chrome Version
$chromePath = "C:\Program Files (x86)\Google\Chrome\Application\chrome.exe"
if (Test-Path $chromePath) {
    $chromeVersion = (Get-Item $chromePath).VersionInfo.FileVersion
}

$chromePath = "C:\Program Files\Google\Chrome\Application\chrome.exe"
```



```
if (Test-Path $chromePath) {  
    $chromeVersion = (Get-Item $chromePath).VersionInfo.FileVersion  
}  
  
$chromeVersion  
pause # or Read-Host
```

Or

```
$chromePaths = @(  
    "C:\Program Files (x86)\Google\Chrome\Application\chrome.exe",  
    "C:\Program Files\Google\Chrome\Application\chrome.exe"  
)  
  
$chromeVersion = $chromePaths | Where-Object { Test-Path $_ } | ForEach-Object  
{ (Get-Item $_).VersionInfo.FileVersion } | Select-Object -First 1  
  
$chromeVersion  
pause # or Read-Host
```

## Return SCCM Agent Version

---

```
# Return SCCM Version  
$sccmPath = "C:\Windows\ccmsetup\ccmsetup.exe"  
if (Test-Path $sccmPath) {  
    $sccmVersion = (Get-Item $sccmPath).VersionInfo.FileVersion  
}  
  
$sccmVersion  
pause # or Read-Host
```



## Detect SCCM Agent

```
-----
# Check if ccmexec.exe is running.
$processName = "ccmexec"
$ccmexecRunning = Get-Process -Name $processName -ErrorAction
'SilentlyContinue'

# Return True if running, False otherwise.
[bool] ($ccmexecRunning -ne $Null)

pause # or Read-Host
```

## Detect FSLogix Environment

```
-----
# Detect FSLogix
$fslogixPath = "C:\Program Files\FSLogix"
$FsLogix = if (Test-Path $fslogixPath) { $true } else { $false }
$FsLogix
pause # or Read-Host
```

Or

```
$fslogixRegistryKey = "HKLM:\SOFTWARE\FSLogix"
$FsLogix = Test-Path $fslogixRegistryKey
$FsLogix
pause # or Read-Host
```

## Detect AzureAd Device

```
-----
# Detect AzureAD
$dsRegStatus = & DSREGCMD /STATUS
$isAzureADJoined = $dsRegStatus -Match "AzureAdJoined\s*:\s*YES"
$azureAD = if ($isAzureADJoined) { $true } else { $false }
$azureAD
pause # or Read-Host
```



## Return Windows Build Version

---

```
# Return Windows Build Version
$osInfo = Get-CimInstance -ClassName Win32_OperatingSystem
$winVersion = "$($osInfo.BuildNumber)" #osInfo.Version - expand build number
$winVersion
pause # or Read-Host
```

## Return BitLocker Passwords

---

### MANAGE-BDE

Run **MANAGE-BDE** command and capture the output.

```
# Return BitLocker Command Line Output
$manageBdeOutput = & MANAGE-BDE -PROTECTORS -GET C: 2>&1

$Lines = $manageBdeOutput -Split "`n"

# RegEx to define the pattern we're looking for.
$passwordPattern = "(\d{6}-\d{6}-\d{6}-\d{6}-\d{6}-\d{6}-\d{6}-\d{6})$"
$Passwords = @()

# Iterate through each line of the BitLocker output, extracting passwords.
ForEach ($Line in $Lines) {
    if ($Line -Match $passwordPattern) {
        # Extract and clean the password.
        $Password = $Matches[1].Trim()
        # Add the password to the array.
        $Passwords += $Password
    }
}

# Create a single string with our BitLocker passwords.
$commaSeparatedPasswords = $Passwords -Join ","

$commaSeparatedPasswords
pause # or Read-Host
```



## Check Status of Service

---

```
$checkService = Get-Service "Application Information" | Select-Object Status  
-ExpandProperty 'Status'  
  
Write-Host $checkService  
  
pause # or Read-Host
```

## Test Certificate Path & Set Certificate Location

---

```
Clear-Host  
  
if (Test-Path 'Cert:\LocalMachine') {  
  
    Write-Host "Set Cert Location to LocalMachine.`n"  
    Set-Location Cert:\LocalMachine  
    Start-Sleep 3  
  
} else {  
  
    Write-Host "Path not detected!`n"  
    Start-Sleep 3  
    break  
}  
  
try {  
  
    Write-Host "Looking for Certificates...`n"  
    Start-Sleep 3  
  
    # Search for the certificate by thumbprint.  
    $cert = Get-ChildItem -Recurse | Where-Object { $_.Thumbprint -eq  
    "e7cc304d588cb38979548d34386a30176f7efcea" }  
    # Or from Subject: $_.Subject -eq "CN=COMMLINK, DC=DOMAIN, DC=COM"  
    # $_.Subject -eq "CN = CodeSign"  
  
    if ($cert) {  
        # Remove the certificate and confirm removal.  
        $cert | Remove-Item -ErrorAction 'Stop'  
        Write-Host "Certificate removed successfully!`n"  
    } else {  
        Write-Host "No certificate found with the specified thumbprint.`n"  
    }  
}
```



```
}

Start-Sleep 3

} catch {

Write-Host "An error occurred: $($_.Exception.Message)"
Start-Sleep 3

}

pause # or Read-Host
```



## Test if Certificate Exists

```
# Microsoft Identity Verification Root Certificate Authority 2020
$certThumbprint = 'f40042e2e5f7e8ef8189fed15519aece42c3bfa2'
if (Test-Path -Path Cert:\LocalMachine\Root\$certThumbprint) {
    Write-Output "Installed"
}

pause # or Read-Host
```

## Delete Certificate Based on Thumbprint

```
# VeriSign Timestamp CA, Expired 2004
$certThumbprint = '18f7c1fcc3090203fd5baa2f861a754976c8dd25'
Get-ChildItem Cert:\LocalMachine\Root | Where-Object { $_.Thumbprint -eq
$certThumbprint} | Remove-Item

pause # or Read-Host
```

## Turn Off Monitors, Sleep Mode

```
Add-Type -TypeDefintion 'using System; using System.Runtime.InteropServices;
public class MonitorControl { [DllImport("user32.dll")] public static extern
int SendMessage(int hWnd, int hMsg, int wParam, int lParam); public static void
TurnOffMonitors() { SendMessage(-1, 0x0112, 0xF170, 2); } }';
[MonitorControl]::TurnOffMonitors()
```



## PowerShell: Interactive Menu

---

```
# Set the window title.
$host.UI.RawUI.WindowTitle = "Interactive PowerShell Greeting Script"

# Set text and background color
$host.UI.RawUI.BackgroundColor = "Black"
$host.UI.RawUI.ForegroundColor = "Green"

# Clear console window.
Clear-Host

# Function to display a greeting.
function Display-Greeting {
    Clear-Host
    Write-Host "=====
    Write-Host "    Welcome to PowerShell    "
    Write-Host "=====`n"

    # Prompt the user for their name.
    $Name = Read-Host "What is your name? "

    # Display a personalized greeting.
    Write-Host "`nHello, $Name! It's great to meet you.`n"

    Write-Host "`nHave a wonderful day, $Name!`n"
}

# Main menu function
function Show-Menu {
    do {
        Clear-Host
        Write-Host "=====
        Write-Host "    Interactive Script Menu    "
        Write-Host "=====`n"
        Write-Host "1. Display a Greeting"
        Write-Host "2. Exit Script`n"
        $choice = Read-Host "Choose an option (1 or 2)"

        switch ($choice) {
            "1" { Display-Greeting; Read-Host "`nPress Enter to return to the menu" }
            "2" { Write-Host "`nGoodbye! Have a great day!"; exit 1 }
        }
    }
}
```



```
        default { Write-Host "`nInvalid choice. Please try again."
                  -ForegroundColor Red; Start-Sleep -Seconds 2 }
    }
} while ($true)

# Start the script.
Show-Menu
```

## Other Ideas

- Display System Uptime
- Show Logged-In Users
- Disk Space Usage Alert
- Network Configuration Summary
- Check for Pending Windows Updates
- CPU and Memory Usage Display
- Show Installed Applications
- List Running Processes
- Check System Event Logs
- Display Active Network Connections
- Automated Backup Reminder
- System Health Check Report
- Display Local Administrator Accounts
- Show System Startup Programs
- Quick Access to Common Admin Tools
- Monitor Antivirus Status
- Check Firewall Settings
- Display Active Services
- Show Recent Software Installations
- Monitor Printer Queue Status
- Automate Temporary File Cleanup
- Track Group Policy Updates
- List Recently Accessed Files
- Generate System Performance Logs
- Identify Disabled Devices
- Schedule System Reboots
- Display Active VPN Connections
- Audit User Login History
- Check BitLocker Encryption Status
- View Shared Folder Permissions



## PowerShell: Random Quotes

---

```
# Set the Window Title
$host.UI.RawUI.WindowTitle = "Interactive PowerShell Quote Script"

# Set Text and Background Color
$host.UI.RawUI.BackgroundColor = "Black"
$host.UI.RawUI.ForegroundColor = "Green"

# Clear Console Window
Clear-Host

# Function to display a random motivational quote
function Get-RandomQuote {

    $quotes = @(
        "Believe you can and you're halfway there. - Theodore Roosevelt",
        "Do what you can, with what you have, where you are. - Theodore Roosevelt",
        "Success is not final, failure is not fatal: It is the courage to continue that counts. - Winston Churchill",
        "The future belongs to those who believe in the beauty of their dreams. - Eleanor Roosevelt",
        "Start where you are. Use what you have. Do what you can. - Arthur Ashe"
    )
    $quotes | Get-Random
}

# Function to display a greeting
function Display-Greeting {

    Clear-Host
    Write-Host "====="
    Write-Host "    Welcome to PowerShell    "
    Write-Host "=====\`n"
    # Display a random motivational quote
    Write-Host "Here's a motivational quote for you: `n"
    Write-Host (Get-RandomQuote) -ForegroundColor Cyan
}

# Main Menu Function
function Show-Menu {
```



```
do {
    Clear-Host
    Write-Host "=====
    Write-Host "    Interactive Script Menu      "
    Write-Host "===== `n"
    Write-Host "1. Display a Quote"
    Write-Host "2. Exit Script`n"
    $choice = Read-Host "Choose an option (1 or 2)"

    switch ($choice) {

        "1" { $Display-Greeting; Read-Host "`nPress Enter to return to the
            menu." }
        "2" { Write-Host "`nGoodbye! Have a great day!"; exit 1 }
        default { Write-Host "`nInvalid choice. Please try again."
            -ForegroundColor Red; Start-Sleep -Seconds 2 }

    }
} while ($true)

# Start the script.
Show-Menu
```

## Other Ideas

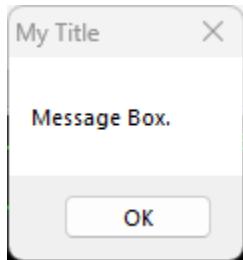
- Daily Tips or Life Hacks
- Joke Generator
- Word of the Day
- Historical Facts
- Programming Snippets
- Wellness Reminders
- Inspirational Images
- Interactive Trivia Quiz
- Random Activity Picker
- Customizable Greeting for Team Members
- Random Password Generator
- Cooking Recipes
- Productivity Prompts
- Study Flashcards
- Random Language Phrases



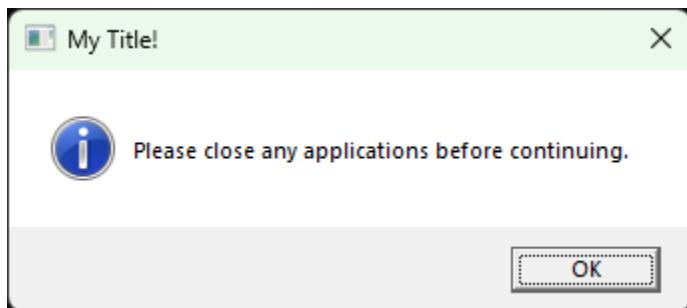
## PowerShell: Message Box

---

```
Add-Type -AssemblyName 'System.Windows.Forms'  
[System.Windows.Forms.MessageBox]::Show("Message Box.", "My Title")
```



```
Add-Type -AssemblyName Microsoft.VisualBasic  
[Microsoft.VisualBasic.Interaction]::MsgBox('Please close any applications  
before continuing.', 'OKOnly, SystemModal, Information', 'My Title')
```



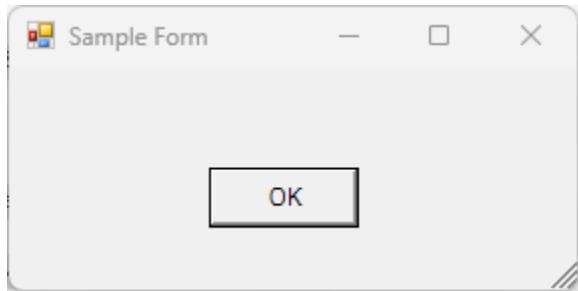


## PowerShell: Forms

---

### Form with OK Button

---



```
# Add necessary assemblies.
Add-Type -AssemblyName System.Windows.Forms
Add-Type -AssemblyName System.Drawing

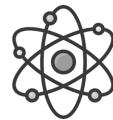
# Create the form.
$Form = New-Object System.Windows.Forms.Form
$Form.Text = "Sample Form"
$Form.Size = New-Object System.Drawing.Size(300, 150)
$Form.StartPosition = "CenterScreen"

# Create the OK button.
$okButton = New-Object System.Windows.Forms.Button
$okButton.Text = "OK"
$okButton.Location = New-Object System.Drawing.Point(100, 50)
$okButton.Size = New-Object System.Drawing.Size(75, 30)

# Add click event for the OK button.
$okButton.Add_Click({
    [System.Windows.Forms.MessageBox]::Show("You clicked OK!", "My Title")
})

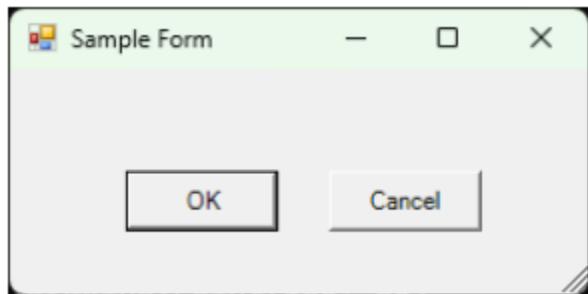
# Add the OK button to the form.
$Form.Controls.Add($okButton)

# Show the form.
$Form.ShowDialog()
```



## Form with OK and Cancel Buttons

---



```
# Add necessary assemblies.
Add-Type -AssemblyName System.Windows.Forms
Add-Type -AssemblyName System.Drawing

# Create the form.
$Form = New-Object System.Windows.Forms.Form
$Form.Text = "Sample Form"
$Form.Size = New-Object System.Drawing.Size(300, 150)
$Form.StartPosition = "CenterScreen"

# Create the OK button.
$okButton = New-Object System.Windows.Forms.Button
$okButton.Text = "OK"
$okButton.Location = New-Object System.Drawing.Point(50, 50)
$okButton.Size = New-Object System.Drawing.Size(75, 30)

# Add click event for the OK button.
$okButton.Add_Click({
    [System.Windows.Forms.MessageBox]::Show("You clicked OK!", "Message")
    # Do stuff here
})

# Create the Cancel button.
$cancelButton = New-Object System.Windows.Forms.Button
$cancelButton.Text = "Cancel"
$cancelButton.Location = New-Object System.Drawing.Point(150, 50)
$cancelButton.Size = New-Object System.Drawing.Size(75, 30)

# Add click event for the Cancel button.
$cancelButton.Add_Click({
    $Form.Close()
})
```



```
# Add buttons to the form.  
$Form.Controls.Add($okButton)  
$Form.Controls.Add($cancelButton)  
  
# Show the form.  
$Form.ShowDialog()
```

## Opacity, Element Opacity

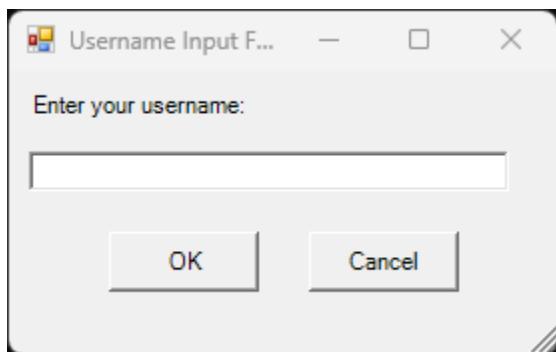
---

```
$Form.Opacity = '.75'
```



## Form with OK, Cancel Buttons & Input Field

---



```
# Add necessary assemblies.
Add-Type -AssemblyName System.Windows.Forms
Add-Type -AssemblyName System.Drawing

# Create the form.
$Form = New-Object System.Windows.Forms.Form
$Form.Text = "Username Input Form"
$Form.Size = New-Object System.Drawing.Size(290, 180)
$Form.StartPosition = "CenterScreen"

# Create a label for the input field.
$usernameLabel = New-Object System.Windows.Forms.Label
$usernameLabel.Text = "Enter your username:"
$usernameLabel.Location = New-Object System.Drawing.Point(10, 10)
$usernameLabel.Size = New-Object System.Drawing.Size(250, 20)

# Create a text box for username input.
$usernameTextBox = New-Object System.Windows.Forms.TextBox
$usernameTextBox.Location = New-Object System.Drawing.Point(10, 40)
$usernameTextBox.Size = New-Object System.Drawing.Size(240, 20)

# Create the OK button.
$okButton = New-Object System.Windows.Forms.Button
$okButton.Text = "OK"
$okButton.Location = New-Object System.Drawing.Point(50, 80)
$okButton.Size = New-Object System.Drawing.Size(75, 30)

# Add click event for the OK button.
$okButton.Add_Click({
    $Username = $usernameTextBox.Text
    if (-not [string]::IsNullOrEmpty($Username)) {
```



```
[System.Windows.Forms.MessageBox]::Show("Your username is: $Username",
"Username")
} else {
[System.Windows.Forms.MessageBox]::Show("Please enter a username.",
"Error")
}
})

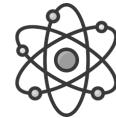
# Create the Cancel button.
$cancelButton = New-Object System.Windows.Forms.Button
$cancelButton.Text = "Cancel"
$cancelButton.Location = New-Object System.Drawing.Point(150, 80)
$cancelButton.Size = New-Object System.Drawing.Size(75, 30)

# Add click event for the Cancel button.
$cancelButton.Add_Click({
    $Form.Close()
})

# Add controls to the form.$Form.Controls.Add($usernameLabel)

$Form.Controls.Add($usernameTextBox)
$Form.Controls.Add($okButton)
$Form.Controls.Add($cancelButton)

# Show the form.
$Form.ShowDialog()
```



## Common Form Options

---

```
$Form = New-Object System.Windows.Forms.Form
$Form.Text = "Form Title"
$Form.Size = New-Object System.Drawing.Size(300, 100)
$Form.StartPosition = "CenterScreen"
$Form.TopMost = $true
$Form.BackColor = "black"
$Form.ControlBox = $false
$Form.FormBorderStyle = "FixedSingle"
$Form.MaximizeBox = $false
$Form.MinimizeBox = $false
$Form.ShowIcon = $false
$Form.ShowInTaskbar = $false
$Form.AcceptButton = $okButton # Button to press Enter
$Form.CancelButton = $cancelButton # Button to press Escape
$Form.Icon =
[System.Drawing.Icon]::ExtractAssociatedIcon('C:\PowerShell\icon.ico')
# $Form.Icon = [System.Drawing.Icon]::ExtractAssociatedIcon((Get-Command
PowerShell.exe).Source)
# You can use icons directly from EXEs. Try this: notepad.exe, calc.exe,
cmd.exe, CERTUTIL.
$Form.AutoSize = $true
$Form.AutoSizeMode = "GrowAndShrink"
$Form.WindowState = "Normal"
$Form.Visible = $true
$Form.HelpButton = $true
$Form.KeyPreview = $true # To capture keypresses globally in the form.
$Form.RightToLeft = "No"
$Form.FormClosing.Add({
    # Custom action on closing.
})
$Form.TopMost = $true
$Form.Padding = New-Object System.Windows.Forms.Padding(5)
$Form.Margin = New-Object System.Windows.Forms.Padding(5)
$Form.MinimumSize = New-Object System.Drawing.Size(200, 100)
$Form.MaximumSize = New-Object System.Drawing.Size(500, 300)
$Form.TextAlign = [System.Drawing.ContentAlignment]::MiddleCenter
$Form.RightToLeftLayout = $true
$Form.AllowDrop = $true
$Form.KeyDown.Add({
    # Capture key presses.
})
$Form.Resize += {
```

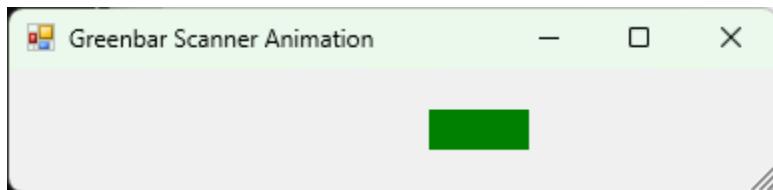


```
# Handle resize events.  
}  
$Form.Shown.Add({  
    # Handle when the form is shown.  
})
```



## Greenbar Animation

---



```
Add-Type -AssemblyName System.Windows.Forms

# Create the form.
$Form = New-Object System.Windows.Forms.Form
$Form.Text = "Greenbar Scanner Animation"
$Form.Size = New-Object System.Drawing.Size(400, 100)
$Form.StartPosition = "CenterScreen"

# Create the green bar.
$greenBar = New-Object System.Windows.Forms.Panel
$greenBar.Size = New-Object System.Drawing.Size(50, 20)
$greenBar.BackColor = 'Green'
$Form.Controls.Add($greenBar)

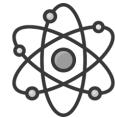
# Set initial position.
$greenBar.Location = New-Object System.Drawing.Point(0,
[math]::Floor(($Form.ClientSize.Height - $greenBar.Height) / 2))

# Create the timer.
$Timer = New-Object System.Windows.Forms.Timer
$Timer.Interval = 50

# Define the animation logic.
$Timer.Add_Tick({
    $greenBar.Left += 5
    if ($greenBar.Left -gt $Form.ClientSize.Width) {
        $greenBar.Left = -$greenBar.Width
    }
})

# Start the animation.
$Timer.Start()

# Show the form.
[void]$Form.ShowDialog()
```



## Green Scanner Animation

---



```
# Add the necessary assemblies.
Add-Type -AssemblyName System.Windows.Forms
Add-Type -AssemblyName System.Drawing

# Create the form.
$Form = New-Object System.Windows.Forms.Form
$Form.Text = "Green Scanner Animation"
$Form.Size = New-Object System.Drawing.Size(300, 100)
$Form.TopMost = $true
$Form.BackColor = "black"
$Form.ControlBox = $false
$Form.FormBorderStyle = "FixedSingle"
$Form.StartPosition = "CenterScreen"

# Create the animation.
$Animation = New-Object System.Windows.Forms.Panel
$Animation.Size = New-Object System.Drawing.Size(10, 20)
$Animation.BackColor = 'lime'

$Form.Controls.Add($Animation)

# Set initial position.
$Animation.Location = New-Object System.Drawing.Point(0,
[math]::Floor(($Form.ClientSize.Height - $Animation.Height) / 2))

# Create the timer.
$Timer1 = New-Object System.Windows.Forms.Timer
$Timer1.Interval = 51

$Timer2 = New-Object System.Windows.Forms.Timer
$Timer2.Interval = 51

$Border = 30
```



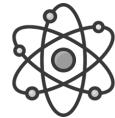
```
# Define the animation logic.
$Timer1.Add_Tick({
    # Moves Right
    $Animation.Left += 10
    if ($Animation.Left -ge ($Form.ClientSize.Width - $Animation.Width -
        $Border)) {
        $Animation.Left = $Form.ClientSize.Width - $Animation.Width - $Border
        $Timer1.Stop()
        $Timer2.Start()
    }
})

$Timer2.Add_Tick({
    # Moves Left
    $Animation.Left -= 10
    if ($Animation.Left -le $Border) {
        $Animation.Left = $Border
        $Timer2.Stop()
        $Timer1.Start()
    }
})

# Add a click event to close the form.
$Form.Add_Click({
    $Timer1.Stop()
    $Timer2.Stop()
    $Form.Close()
})

# Start the animation.
$Timer1.Start()

# Show the form.
[void]$Form.ShowDialog()
```



## Red Scanner Animation

---



```
# Add the necessary assemblies.
Add-Type -AssemblyName System.Windows.Forms
Add-Type -AssemblyName System.Drawing

# Enable double buffering for the form and panel
$DoubleBufferedForm = [System.Windows.Forms.Form]::GetProperty("DoubleBuffered",
[System.Reflection.BindingFlags] "NonPublic, Instance")
$DoubleBufferedPanel =
[System.Windows.Forms.Control]::GetProperty("DoubleBuffered",
[System.Reflection.BindingFlags] "NonPublic, Instance")

# Create the form.
$form = New-Object System.Windows.Forms.Form
$form.Text = "Red Scanner Animation"
$form.Size = New-Object System.Drawing.Size(300, 0)
$form.TopMost = $true
$form.BackColor = "black"
$form.ControlBox = $false
$form.FormBorderStyle = "None"
$form.StartPosition = "CenterScreen"

# Enable double buffering on the form
$DoubleBufferedForm.SetValue($form, $true)

# Create the graphic.
$Animation = New-Object System.Windows.Forms.Panel
$AnimationW = 25
$Animation.Size = New-Object System.Drawing.Size($AnimationW, 45)
$Animation.BackColor = 'red'

# Enable double buffering on the panel
$DoubleBufferedPanel.SetValue($Animation, $true)

$form.Controls.Add($Animation)

# Set initial position.
$Animation.Location = New-Object System.Drawing.Point(0,
[Math]::Floor(($form.ClientSize.Height - $Animation.Height) / 2 + 16))
$animationMovement = 10
```



```
$Border = 1
$timerInterval = 40

# Create the timer.
$Timer1 = New-Object System.Windows.Forms.Timer
$Timer1.Interval = $timerInterval

$Timer2 = New-Object System.Windows.Forms.Timer
$Timer2.Interval = $timerInterval

# Define the animation logic.
$Timer1.Add_Tick({
    # Moves Right
    $Animation.Left += $animationMovement
    if ($Animation.Left -ge ($Form.ClientSize.Width - $Animation.Width -
$Border)) {
        $Animation.Left = $Form.ClientSize.Width - $Animation.Width - $Border
        $Timer1.Stop()
        $Timer2.Start()
    }
})

$Timer2.Add_Tick({
    # Moves Left
    $Animation.Left -= $animationMovement
    if ($Animation.Left -le $Border) {
        $Animation.Left = $Border
        $Timer2.Stop()
        $Timer1.Start()
    }
})

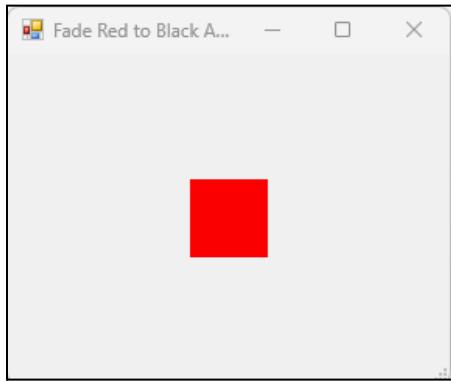
# Add a click event to close the form.
$Form.Add_Click({
    $Timer1.Stop()
    $Timer2.Stop()
    $Form.Close()
})

# Start the animation.
$Timer1.Start()

# Show the form.
[void]$Form.ShowDialog()
```

## Fade Red to Black Animation

---



```
# Add the necessary assemblies.
Add-Type -AssemblyName System.Windows.Forms
Add-Type -AssemblyName System.Drawing

# Enable double buffering for the form and panel
$DoubleBufferedForm = [System.Windows.Forms.Form]::GetProperty("DoubleBuffered",
[System.Reflection.BindingFlags] "NonPublic, Instance")

$DoubleBufferedPanel =
[System.Windows.Forms.Control]::GetProperty("DoubleBuffered",
[System.Reflection.BindingFlags] "NonPublic, Instance")

# Create the form.
$form = New-Object System.Windows.Forms.Form
$form.Text = "Fade Red to Black Animation"
$form.Size = New-Object System.Drawing.Size(300, 250)
$form.StartPosition = "CenterScreen"

# Enable double buffering on the form
$DoubleBufferedForm.SetValue($form, $true)

# Add a label to the form.
$form.Controls.Add($label)

$global:Timer = New-Object System.Windows.Forms.Timer
$global:Timer.Interval = 20
$global:Counter = 255 # Starting Count

# Create the block.
$block = New-Object System.Windows.Forms.Panel
```



```
$Block.Size = New-Object System.Drawing.Size(50, 50)
$Block.BackColor = [System.Drawing.Color]::FromArgb(255, 0, 0)
$Block.Location = New-Object System.Drawing.Point(((($Form.ClientSize.Width -
$Block.Width) / 2), ((($Form.ClientSize.Height - $Block.Height) / 2)))
$form.Controls.Add($Block)

# Add the Tick event to the timer.
$global:Timer.Add_Tick({
    $global:Counter--

    $Block.BackColor = [System.Drawing.Color]::FromArgb($global:Counter--, 0,
    0)

    if ($global:Counter -le 0) {
        $global:Counter = 255

    }
})

# Add a click event to close the form.
$form.Add_Click({
    $global:Timer.Stop()
    $global:Counter = 255
    $form.Close()
})

# Start the timer and show the form.
$global:Timer.Start()
$form.ShowDialog()
```



## Fade In & Out Animation

---



```
# Add the necessary assemblies.
Add-Type -AssemblyName System.Windows.Forms
Add-Type -AssemblyName System.Drawing

# Enable double buffering for the form and panel
$DoubleBufferedForm = [System.Windows.Forms.Form]::GetProperty("DoubleBuffered",
[System.Reflection.BindingFlags] "NonPublic, Instance")

# Create the form.
$form = New-Object System.Windows.Forms.Form
$form.Text = "Red Scanner Animation"
$form.Size = New-Object System.Drawing.Size(50, 0)
$form.TopMost = $true
$form.BackColor = "black"
$form.ControlBox = $false
$form.FormBorderStyle = "None"
$form.StartPosition = "CenterScreen"

# Enable double buffering on the form.
$DoubleBufferedForm.SetValue($form, $true)

# Timer and variables for fading effect.
$global:Timer = New-Object System.Windows.Forms.Timer
$global:Timer.Interval = 1
$global:Counter = 255 # Starting red value
# Fading direction (True = red to black. False = black to red.)
$global:FadingOut = $true

# Create the block.
$block = New-Object System.Windows.Forms.Panel
$block.Size = New-Object System.Drawing.Size(100, 70)
$block.BackColor = [System.Drawing.Color]::FromArgb($global:Counter, 0, 0)
$block.Location = New-Object System.Drawing.Point(((($form.ClientSize.Width -
$block.Width +15 ) / 2), ((($form.ClientSize.Height - $block.Height +5) / 2)))
$form.Controls.Add($block)

# Add the Tick event to the timer.
```



```
$global:Timer.Add_Tick({
    if ($global:FadingOut) {
        $global:Counter-- # Decrease red value.
        if ($global:Counter -le 0) {
            $global:Counter = 0
            $global:FadingOut = $false # Switch to fading in.
        }
    } else {
        $global:Counter++ # Increase red value.
        if ($global:Counter -ge 255) {
            $global:Counter = 255
            $global:FadingOut = $true # Switch to fading out.
        }
    }

    # Update the block's color.
    $Block.BackColor = [System.Drawing.Color]::FromArgb($global:Counter, 0, 0)
})

# Add a click event to close the form.
$form.Add_Click({
    $global:Timer.Stop()
    $Form.Close()
})

# Start the timer and show the form.
$global:Timer.Start()
$form.ShowDialog()
```



## Countdown 10-1

---



```
# Add the necessary assemblies.
Add-Type -AssemblyName System.Windows.Forms
Add-Type -AssemblyName System.Drawing

# Create the form.
$Form = New-Object System.Windows.Forms.Form
$Form.Text = "Countdown Timer"
$Form.Size = New-Object System.Drawing.Size(300, 150)
$Form.StartPosition = "CenterScreen"

$Label = New-Object System.Windows.Forms.Label
$Label.Text = "10"
$Label.Font = New-Object System.Drawing.Font("Arial", 24)
$Label.Size = New-Object System.Drawing.Size(100, 50)
$Label.Location = New-Object System.Drawing.Point(100, 40)
$Label.TextAlign = [System.Drawing.ContentAlignment]::MiddleCenter

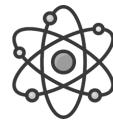
# Add a label to the form.
$Form.Controls.Add($Label)

$global:Timer = New-Object System.Windows.Forms.Timer
$global:Timer.Interval = 1000 # 1 Second Interval.
$global:Counter = 10 # Starting Count

# Add the Tick event to the timer.
$global:Timer.Add_Tick({
    $global:Counter--
    $Label.Text = $global:Counter.ToString()
    if ($global:Counter -le 0) {
        $global:Timer.Stop()
        $Label.Text = "Done!"
        Start-Sleep 5
    }
})
```

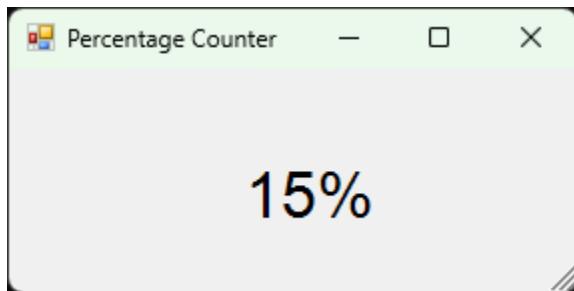


```
$global:Timer.Dispose()  
$Form.Close()  
}  
})  
  
# Start the timer.  
$global:Timer.Start()  
  
# Show the form.  
$Form.ShowDialog()
```



## Percentage Counter 1-100%

---



```
# Add necessary assemblies.
Add-Type -AssemblyName System.Windows.Forms
Add-Type -AssemblyName System.Drawing

# Create the form.
$Form = New-Object System.Windows.Forms.Form
$Form.Text = "Percentage Counter"
$Form.Size = New-Object System.Drawing.Size(300, 150)
$Form.StartPosition = "CenterScreen"

$Label = New-Object System.Windows.Forms.Label
$Label.Text = "1%"
$Label.Font = New-Object System.Drawing.Font("Arial", 24)
$Label.Size = New-Object System.Drawing.Size(100, 50)
$Label.Location = New-Object System.Drawing.Point(100, 40)
$Label.TextAlign = [System.Drawing.ContentAlignment]::MiddleCenter

# Add a label to the form.
$Form.Controls.Add($Label)

$global:Timer = New-Object System.Windows.Forms.Timer
$global:Timer.Interval = 500 # Half Second Interval
$global:Counter = 1 # Starting Count

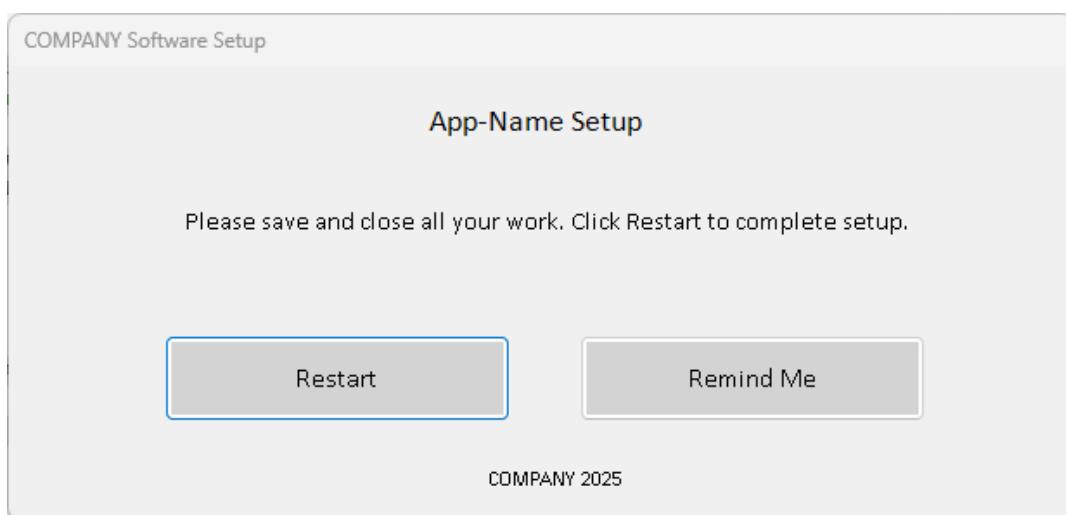
# Add the Tick event to the timer.
$global:Timer.Add_Tick({
    $global:Counter++
    $Label.Text = $global:Counter.ToString() + "%"
    if ($global:Counter -eq 100) {
        $Label.Text = "Done!"
        $global:Timer.Stop()
        Start-Sleep 5
    }
})
```



```
$global:Timer.Dispose()  
$Form.Close()  
}  
})  
  
# Start the timer.  
$global:Timer.Start()  
  
# Show the form.  
$Form.ShowDialog()
```



## Empty Reboot Form



```
# Load Main Function
function Start-Reboot {
    $global:Counter = 1
    $ErrorActionPreference = 'SilentlyContinue'

    # Play Alert Sound
    $Sound = New-Object System.Media.SoundPlayer;
    $Sound.SoundLocation = "C:\Windows\Media\notify.wav";
    $Sound.PlayLooping();
    $Flag = $false;

    1..10 | ForEach {
        if ($_ -gt 1) {$Flag = $true} else {Start-Sleep 1}
        if ($Flag) {$Sound.Stop() }
    }

    # Load Assemblies
    Add-Type -AssemblyName PresentationFramework
    Add-Type -AssemblyName System.Windows.Forms
    Add-Type -AssemblyName System.Drawing
    [System.Windows.Forms.Application]::EnableVisualStyles()

    # Instantiate Objects
    $Form = New-Object System.Windows.Forms.Form
```



```
$Label1 = New-Object System.Windows.Forms.Label
$Label2 = New-Object System.Windows.Forms.Label
$Label3 = New-Object System.Windows.Forms.Label
$Button1 = New-Object System.Windows.Forms.Button
$Button2 = New-Object System.Windows.Forms.Button
$InitialFormWindowState = New-Object System.Windows.Forms.FormWindowState

$Form_StateCorrection_Load =
{
    $Form.WindowState = $InitialFormWindowState
}

# Main Form
# $Form.Icon = [Drawing.Icon]::ExtractAssociatedIcon((Get-Command
# PowerShell).Path)
$Form.Controls.Add($Button1)
$Form.Controls.Add($Button2)
$Form.Controls.Add($Label1)
$Form.Controls.Add($Label2)
$Form.Controls.Add($Label3)
$Form.AutoScaleDimensions = '8, 17'
$Form.AutoScaleMode = 'Font'
$Form.ClientSize = '616, 262'
$Form.FormBorderStyle = 'FixedDialog'
$Form.Margin = '5, 5, 5, 5'
$Form.MaximizeBox = $false
$Form.MinimizeBox = $false
$Form.ControlBox = $false
$Form.Name = 'App-Name'
$Form.StartPosition = 'CenterScreen'
$Form.Text = 'COMPANY Software Setup'
$Form.ShowInTaskbar = $false
$Form.TopMost = $true # Force the window to stay on top.
$Form.Add_Load($Form_Load)

# Label1
$Label1.Location = New-Object System.Drawing.Point(240,20)
$Label1.Font = 'Calibri, 13.25pt'
$Label1.Size = New-Object System.Drawing.Size(550,50)
$Label1.Text = 'App-Name Setup'

# Label2
$Label2.Location = New-Object System.Drawing.Point(100,80)
$Label2.Font = 'Calibri, 11.25pt'

$Label2.Size = New-Object System.Drawing.Size(440,50)
```



```
$Label2.Text = 'Please save and close all your work. Click Restart to complete setup.'  
  
# Label3  
$Label3.Location = New-Object System.Drawing.Point(275,230)  
$Label3.Font = 'Calibri, 9.25pt'  
$Label3.Size = New-Object System.Drawing.Size(100,80)  
$Label3.Text = 'COMPANY 2025'  
  
# Button  
$Button1.Font = 'Calibri, 12.25pt'  
$Button1.DialogResult = 'OK'  
$Button1.Location = '90, 155'  
$Button1.Margin = '5, 5, 5, 5'  
$Button1.Name = 'buttonOK1'  
$Button1.Size = '200, 50'  
$Button1.BackColor = "LightGray"  
$Button1.ForeColor = "black"  
$Button1.Text = '&Restart'  
$Button1.UseCompatibleTextRendering = $true  
$Button1.UseVisualStyleBackColor = $false  
  
# DISABLED  
# $Button1.Add_Click({$Form.Add_FormClosing({$_.Cancel=$false});  
#     Restart-Computer -Force})  
  
$Button1.Add_Click({$Form.Add_FormClosing({$_.Cancel=$false});})  
$Button1.TabIndex = 2  
  
# Button  
$Button2.Font = 'Calibri, 12.25pt'  
$Button2.DialogResult = 'OK'  
$Button2.Location = '330, 155'  
$Button2.Margin = '5, 5, 5, 5'  
$Button2.Name = 'buttonOK2'  
$Button2.Size = '200, 50'  
$Button2.BackColor = "LightGray"  
$Button2.ForeColor = "black"  
$Button2.Text = '&Remind Me'  
$Button2.UseCompatibleTextRendering = $true  
$Button2.UseVisualStyleBackColor = $false  
$Button2.Add_Click($Button2_click)  
$Button2.TabIndex = 3  
  
# Form Config  
$InitialFormWindowState = $Form.WindowState
```



```
$Form.Add_Load($Form_StateCorrection_Load)

# Focus Process
$WindowState = '[DllImport("user32.dll")] public static extern bool
ShowWindow(int handle, int state);'

Add-Type -Name win -Member $WindowState -Namespace native
[native.win]::ShowWindow(([System.Diagnostics.Process]::GetCurrentProcess()
| Get-Process).MainWindowHandle, 0)
SetForegroundWindow(this.Handle)

# Show Form
$Form.Add_FormClosing({$_.Cancel=$true})
[void] $Form.ShowDialog()

$Counter
}

$Button2_click ={

if ($Counter -lt 3) {

    $script:counter++

    $Form.WindowState = [System.Windows.Forms.FormWindowState]::Minimized
    $Form.Hide()

    # Start-Sleep -Seconds 30 # for testing purposes only
    Start-Sleep -Seconds 3600 #1 hour

    # I removed this function. It was meant to be tried 3 times before
    # forcing an upgrade action.
    # If counter equals 3, perform an upgrade.
    # CheckCounter

    $Form.Show()
    $Form.WindowState = [System.Windows.Forms.FormWindowState]::Normal
}

function CloseForm {
    $Form.Close()
}

# Run Main Function
```



```
$global:Counter = 1
```

```
Start-Reboot
```

## Other Ideas

- Automated Workstation Reboots
- User Notification System
- Disk Cleanup Prompts
- Software Installation Checker
- Policy Compliance Reminder
- Update Status Dashboard
- Controlled Shutdown Options
- Session Countdown Timer
- Custom Branding for IT Operations
- Interactive Troubleshooting Tool
- Remote Support Request Form
- Password Expiration Reminder
- Pending Approval for Software Installation
- Automated Log Collection Interface
- Scheduled Backup Confirmation
- Hardware Resource Monitoring Dashboard
- System Lock Notification
- Network Connectivity Test Interface
- Temporary Admin Access Request
- User Feedback Collection Form



## PowerShell: Task Scheduler

---

How to test from the System Account ([PsExec](#)):

```
PsExec64 -S CMD  
PowerShell -ExecutionPolicy Bypass -File 'C:\PowerShell\script.ps1'
```

### On Demand App Install

---

**Scope:** Admin, System Account, PC, VDI

Using PowerShell, install enterprise apps using a scheduled task. Create a silent install package, or pass the silent install options to a local setup file. Set the permissions on the task using ICACLS. Once this script has been deployed, and the task has been created, users can install apps as System. The processing of this code is similar to the Batch script version, but uses cmdlets, rather than SCHTASK.exe.

```
$MSIFile = "C:\PowerShell\setup.msi"  
$TaskName = "App-Install"  
$TaskPath = "\  
  
$TaskAction = New-ScheduledTaskAction -Execute "msiexec.exe" -Argument "/QN /I`"$MSIFile`"""  
  
$TaskPrincipal = New-ScheduledTaskPrincipal -UserId "SYSTEM" -LogonType ServiceAccount  
  
$TaskSettings = New-ScheduledTaskSettingsSet -AllowStartIfOnBatteries -DontStopIfGoingOnBatteries -StartWhenAvailable  
  
$TaskTrigger = New-ScheduledTaskTrigger -Once -At "00:00"  
  
# Register the task.  
Register-ScheduledTask -TaskName $TaskName -TaskPath $TaskPath -Action $TaskAction -Principal $TaskPrincipal -Settings $TaskSettings -Trigger $TaskTrigger -Force  
  
# Set permissions to allow all users to run it.  
$TaskFile = "C:\Windows\System32\Tasks\$TaskName"  
CMD "/C ICACLS $TaskFile /Grant Everyone:(RX)" | Out-Null
```



## On Demand App Repair

### ICACLS

Repair enterprise apps using a scheduled task. Create a silent repair package, or pass the silent repair options to a local setup file. Set the permissions on the task using ICACLS. Once this script has been deployed, and the task has been created, users can repair apps as System.

```
$MSIFile = "C:\PowerShell\setup.msi"
$TaskName = "App-Repair"
$TaskPath = "\"

$TaskAction = New-ScheduledTaskAction -Execute "msiexec.exe" -Argument "/QN
/FAUMS `"$MSIFile`"""

$TaskPrincipal = New-ScheduledTaskPrincipal -UserId "SYSTEM" -LogonType
ServiceAccount

$TaskSettings = New-ScheduledTaskSettingsSet -AllowStartIfOnBatteries
-DontStopIfGoingOnBatteries -StartWhenAvailable

$TaskTrigger = New-ScheduledTaskTrigger -Once -At "00:00"

# Register the task.
Register-ScheduledTask -TaskName $TaskName -TaskPath $TaskPath -Action
$TaskAction -Principal $TaskPrincipal -Settings $TaskSettings -Trigger
$TaskTrigger -Force

# Set permissions to allow all users to run it.
$TaskFile = "C:\Windows\System32\Tasks\$TaskName"
CMD "/C ICACLS $TaskFile /Grant Everyone:(RX)" | Out-Null
```



## On Demand App Download

### ICAcls

**Scope:** Admin, System Account, PC, VDI

Download enterprise apps using a scheduled task. Set the permissions on the task using ICAcls. Once this script has been deployed, and the task has been created, users can download apps as System.

```
# Define the path to an online MSI file
$DownloadFile =
"https://dl.google.com/tag/s/dl/chrome/install/googlechromestandaloneenterprise
64.msi"

# Define the name of the scheduled task
$TaskName = "App-Download"

# Create the scheduled task to run the download command as SYSTEM
$TaskAction = New-ScheduledTaskAction -Execute "curl.exe" -Argument
`"$DownloadFile`" --output C:\PowerShell\setup.msi

$TaskPrincipal = New-ScheduledTaskPrincipal -UserId "SYSTEM" -LogonType
ServiceAccount

$TaskTrigger = New-ScheduledTaskTrigger -Once -At "00:00"

$TaskSettings = New-ScheduledTaskSettingsSet -AllowStartIfOnBatteries
-DontStopIfGoingOnBatteries -StartWhenAvailable

# Register the task
Register-ScheduledTask -TaskName $TaskName -Action $TaskAction -Principal
$TaskPrincipal -Trigger $TaskTrigger -Settings $TaskSettings -Force

# Set permissions to allow all users to run it.
$TaskFile = "C:\Windows\System32\Tasks\$TaskName"
CMD "/C ICACLS $TaskFile /Grant Everyone:(RX)" | Out-Null
```



## Create Scheduled Task to Run Process or Script

---

```
# Define the path to the PowerShell script.  
# This could also be a path to a setup file or package.  
$pshScript = "C:\PowerShell\script.ps1"  
  
# Define the name of the scheduled task  
$TaskName = "Run-PowerShell-Script"  
  
# Define the task action.  
$TaskAction = New-ScheduledTaskAction -Execute "powershell.exe" -Argument  
"-NoProfile -ExecutionPolicy Bypass -File `"$pshScript`""  
  
$TaskPrincipal = New-ScheduledTaskPrincipal -UserId "SYSTEM" -LogonType  
ServiceAccount  
  
$TaskTrigger = New-ScheduledTaskTrigger -Once -At "00:00"  
  
$TaskSettings = New-ScheduledTaskSettingsSet -AllowStartIfOnBatteries  
-DontStopIfGoingOnBatteries -StartWhenAvailable  
  
# Register the task  
Register-ScheduledTask -TaskName $TaskName -Action $TaskAction -Principal  
$TaskPrincipal -Trigger $TaskTrigger -Settings $TaskSettings -Force  
  
# Set permissions to allow all users to run it.  
$TaskFile = "C:\Windows\System32\Tasks\$TaskName"  
CMD "/C ICACLS $TaskFile /Grant Everyone:(RX)" | Out-Null
```



## 7. AutoHotKey

---

### Brief Overview

**AutoHotKey** burst onto the scene in 2003 with a simple yet ambitious goal: to make automation on Windows accessible to everyone. Whether you want to remap a key, automate repetitive tasks, or build a custom workflow, AutoHotKey (AHK) is like having your own digital assistant, always ready to streamline your day and improve efficiency.

With its approachable and highly flexible syntax, AutoHotKey enables beginners to dip their toes into scripting while offering advanced users the power to craft complex, tailored solutions. Want to create hotkeys that save you from endless clicks? Automate tedious file management processes? Control applications or windows with precision and ease? AutoHotKey handles it all—and then some. Its versatility extends to gamers, software testers, and anyone who values squeezing maximum productivity out of their system. Whether it's setting up macros for your favorite game, enhancing your testing workflow, or simply managing daily tasks, AHK rises to the challenge.

Even as newer automation tools have entered the scene, AHK continues to hold its ground, offering a near-perfect blend of simplicity, speed, and sheer customizability. From creating quick shortcuts that save seconds to building fully-fledged standalone applications, AutoHotKey proves that sometimes, the simplest tools can be the most powerful and enduring.



## Why is it still practical today?

- **Hotkeys and Hotstrings:** AutoHotKey is renowned for its ability to create custom hotkeys and hotstrings, allowing users to trigger commands with keyboard shortcuts or abbreviations, significantly improving efficiency.
- **GUI Automation:** Like AutoIt, AutoHotKey excels in automating tasks that involve interacting with graphical user interfaces, making it perfect for automating programs, games, or testing environments.
- **Ease of Use:** The syntax is simple and intuitive, which makes AutoHotKey a great starting point for beginners while still providing powerful features for advanced users.
- **Customizable Scripts:** AutoHotKey allows users to create highly customized automation solutions. Scripts can range from simple keyboard shortcuts to complex applications that can automate entire workflows.
- **Lightweight and Fast:** AHK scripts are lightweight and execute quickly, making them ideal for tasks that require rapid execution or need to be triggered frequently.
- **Active Community:** The AutoHotKey community is large and active, offering a wealth of resources such as forums, tutorials, and user-generated scripts, which makes learning and troubleshooting easier.

Even though there are more complex automation tools, AutoHotKey remains a popular and practical choice for quick, flexible automation on Windows. Its combination of simplicity, powerful features, and an active user community ensures its continued relevance in the world of automation and scripting.

When naming your AutoHotKey scripts, use the .ahk extension.

**Examples:** Script.ahk, My\_Script.ahk, Copy-Script.ahk



## AutoHotKey: Design

---

<https://www.autohotkey.com>

### Basic Construction

---

```
; AutoHotKey Greeting Script

; Disable menu display when the script is running.
#NoTrayIcon

; Set a window title (used for InputBox or other windows).
Gui, +AlwaysOnTop -Caption +ToolWindow ; Settings for the GUI window.

; Main script starts here.
; Prompt the user for their name.
InputBox, UserName, Greeting Script, Please enter your name:,,400,200

; Check if the user entered a name or canceled.
if (ErrorLevel)
{
    MsgBox, 48, Greeting Script, You canceled the script. Goodbye!
    ExitApp
}

; Display a personalized greeting.
MsgBox, 64, Greeting Script, Hello, %UserName%! Have a great day!

; Exit the script.
ExitApp
```



## AutoHotKey: Mouse Block

---

```
#NoTrayIcon  
BlockInput, MouseMove  
Sleep 30000  
BlockInput, MouseMoveOff  
Suspend
```

## AutoHotKey: Volume Control with Mouse Wheel

---

```
#NoTrayIcon  
  
!WheelUp:: ; Alt + Mouse Wheel Up  
    SoundSet, +5 ; Increase volume by 5%  
    return  
  
!WheelDown:: ; Alt + Mouse Wheel Down  
    SoundSet, -5 ; Decrease volume by 5%  
    return
```

## AutoHotKey: Window Always on Top

---

```
#NoTrayIcon  
  
^Space:: ; Ctrl + Space  
    WinSet, AlwaysOnTop, Toggle, A ; A means the active window.  
    return
```



## AutoHotKey: Caps Lock to Toggle Mute

---

```
#NoTrayIcon

CapsLock::
    SoundSet, +1, , mute ; Toggle mute
    return
```

## AutoHotKey: Quick Launch Applications

---

```
#NoTrayIcon

^!C:: ; Ctrl + Alt + C
    Run, calc.exe ; Launch Calculator
    return

^!N:: ; Ctrl + Alt + N
    Run, notepad.exe ; Launch Notepad
    return
```

## AutoHotKey: Play a Tune

---

```
#NoTrayIcon

; Play "Mary Had a Little Lamb"
^!P:: ; Ctrl + Alt + P to play the tune
    ; Notes: E D C D E E E, D D D, E G G
    SoundBeep, 659, 300 ; E
    SoundBeep, 587, 300 ; D
    SoundBeep, 523, 300 ; C
    SoundBeep, 587, 300 ; D
    SoundBeep, 659, 300 ; E
    SoundBeep, 659, 300 ; E
    SoundBeep, 659, 600 ; E (hold)

    SoundBeep, 587, 300 ; D
    SoundBeep, 587, 300 ; D
    SoundBeep, 587, 600 ; D (hold)
```



```
SoundBeep, 659, 300 ; E  
SoundBeep, 784, 300 ; G  
SoundBeep, 784, 600 ; G (hold)  
return
```



## 8. Autolt

---

### Brief Overview

**Autolt** is the quiet powerhouse of Windows automation, purpose-built to make interacting with the GUI and performing scripting tasks effortless. Launched in 1999, Autolt was created with one clear goal in mind: to eliminate the monotony of repetitive tasks and empower users to take full control of their Windows environment. Over the years, it has become a go-to solution for anyone looking to enhance productivity and streamline workflows.

Featuring a scripting language inspired by BASIC, Autolt is remarkably user-friendly and easy to pick up, even for beginners with no prior programming experience. It enables users to craft scripts that automate everything from simple mouse clicks and keyboard inputs to advanced file operations and system-level tasks. With its intuitive syntax and extensive capabilities, Autolt bridges the gap between simplicity and power, making it equally suitable for quick fixes and complex automation projects.

As the tool has evolved, so has its functionality. Autolt now supports creating standalone executable files, allowing users to distribute their scripts as fully independent applications. This feature has cemented Autolt's reputation as a versatile and reliable tool for both casual users and seasoned IT professionals. Whether you're building intricate GUI interactions, managing extensive file operations, or developing custom applications, Autolt provides the flexibility and reliability needed to get the job done.

From hobbyists exploring the world of automation to professionals optimizing large-scale workflows, Autolt has earned its place as a trusted and enduring tool. Its unique blend of simplicity, flexibility, and Windows-specific focus ensures that it remains a powerful ally for tackling automation challenges of any size.



## Why is it still practical today?

- **Extensive GUI Automation:** AutoIt excels in automating tasks that require interacting with graphical user interfaces, making it ideal for automating desktop applications, installation processes, or testing workflows.
- **Ease of Use:** Its simple syntax, based on BASIC, makes it easy to learn for those new to scripting and programming, while still offering powerful features for more advanced users.
- **Standalone Executables:** AutoIt scripts can be compiled into standalone executable files (.exe), allowing for easy distribution and execution on machines without requiring AutoIt to be installed.
- **Robust Windows Integration:** AutoIt can automate a wide range of tasks on Windows, such as file management, system settings, and application control, making it a versatile tool for administrators and developers.
- **Active Community and Support:** AutoIt has an active user community, offering plenty of resources, including tutorials, forums, and libraries, which make it easy to get started and find solutions to common problems.

Even though there are newer automation tools emerging, AutoIt remains popular for its robust GUI automation capabilities, simplicity, and flexibility. Whether used for personal productivity, software testing, or system administration, AutoIt continues to be a practical and effective choice for automating repetitive tasks on Windows.

When naming your AutoIt scripts, use the .au3 extension.

**Examples:** Script.ahk, My\_Script.ahk, Copy-Script.ahk



## AutoIt: Design

---

<https://www.autoitscript.com/>

### Basic Construction

---

Here we are; here we go. Let's build our first **AutoIt** script. Using your favorite code editor (try out Notepad++ or the AutoIt editor), type and then save this code as `script.au3`. Double-click the script to *launch* it.

```
; AutoIt Greeting Script

; Disable the display of the AutoIt tray icon.
Opt("TrayIconHide", 1)

; Prompt the user for their name using an input box.
Local $name = InputBox("Greeting Script", "Please enter your name:", "", "", 400, 200)

; Check if the user pressed Cancel or left the input empty.
If @error Then
    MsgBox(48, "Greeting Script", "You canceled the script. Goodbye!")
    Exit
EndIf

; Display a personalized greeting.
MsgBox(64, "Greeting Script", "Hello, " & $name & "! Have a great day!")

; Exit the script.
Exit
```



## Create Text File

```
-----  
;  
Create Text File  
FileWrite("C:\\\\AutoIt\\\\File.txt", "")  
  
MsgBox(0, "Notification", "Text file created!")
```

## Create Folder

```
-----  
;  
Create Folder  
DirCreate("C:\\\\AutoIt\\\\Backup")  
  
MsgBox(0, "Notification", "Backup folder created!")
```

## Copy File

```
-----  
;  
Create File  
FileWrite("C:\\\\AutoIt\\\\File.txt", "")  
  
;  
Create Folder  
DirCreate("C:\\\\AutoIt\\\\Backup")  
  
;  
Copy file to the folder.  
FileCopy("C:\\\\AutoIt\\\\File.txt", "C:\\\\AutoIt\\\\Backup\\\\", 1)  
  
MsgBox(0, "Notification", "File.txt copied to C:\\AutoIt\\Backup")
```

## Copy Folder

```
-----  
;  
Create Folder1  
DirCreate("C:\\\\AutoIt\\\\Folder1")  
  
;  
Create Folder2  
DirCreate("C:\\\\AutoIt\\\\Folder2")
```



```
; Create File.txt  
FileWrite("C:\\AutoIt\\Folder1\\File.txt", "")  
  
; Copy Folder1 to Folder2.  
DirCopy("C:\\AutoIt\\Folder1", "C:\\AutoIt\\Folder2", 1)  
  
MsgBox(0, "Notification", "Folder1 copied to C:\\AutoIt\\Folder2")
```

## Delete File

---

```
; Delete File  
FileDelete("C:\\AutoIt\\Test-File.txt")  
  
MsgBox(0, "Notification", "Test-File.txt has been deleted!")
```

## Delete Folder

---

```
; Create Test-Folder  
DirCreate("C:\\AutoIt\\Test-Folder")  
  
; Delete Test-Folder  
DirRemove("C:\\AutoIt\\Test-Folder", 1)  
  
MsgBox(0, "Notification", "Test-Folder has been deleted!")
```

## Open Notepad, Type 'Test'

---

```
; Open Notepad  
Run("notepad.exe")  
  
; Wait for Notepad to become active.  
WinWaitActive("Untitled - Notepad")  
  
; Type "test" into Notepad.  
Send("Test")
```



```
; Get the process ID (PID) for Notepad.  
Local $sPid = ProcessExists("notepad.exe")  
  
; Check if Notepad is running.  
If $sPid = 0 Then  
    Exit  
EndIf
```



## AutoIt: Move Window Down

---

```
; Get the process ID (PID) for Notepad
Local $sPid = ProcessExists("notepad.exe")

; Check if Notepad is running.
If $sPid = 0 Then
    Exit
EndIf

; Find the window handle for Notepad.
Local $hWnd = WinGetHandle("[CLASS:Notepad]")

; Check if the window handle is valid.
If $hWnd = "" Then
    Exit
EndIf

; Retrieve the window title.
Local $WindowTitle = WinGetTitle($hWnd)

; Ensure the window is active.
WinWaitActive($WindowTitle, "", 1)

; Bring the window into focus
WinActivate($hWnd)

; Get current window dimensions.
Local $aPos = WinGetPos($hWnd)
If @error Then
    Exit
EndIf

; Move the window slightly down.
Local $iNewX = $aPos[0]
Local $iNewY = $aPos[1] + 10

; Reposition the window (you can leave the width/height unchanged)
WinMove($WindowTitle, "", $iNewX, $iNewY)

; Optionally, focus the window again (to ensure it is in the Foreground)
WinActivate($hWnd)
```



## AutoIt: Move Window Up

---

```
; Get the process ID (PID) for Notepad.  
Local $sPid = ProcessExists("notepad.exe")  
  
; Check if Notepad is running  
If $sPid = 0 Then  
    Exit  
EndIf  
  
; Find the window handle for Notepad.  
Local $hWnd = WinGetHandle("[CLASS:Notepad]")  
  
; Check if the window handle is valid  
If $hWnd = "" Then  
    Exit  
EndIf  
  
; Retrieve the window title.  
Local $winTitle = WinGetTitle($hWnd)  
  
; Ensure the window is active  
WinWaitActive($winTitle, "", 1)  
  
; Bring the window into focus.  
WinActivate($hWnd)  
  
; Get current window dimensions.  
Local $aPos = WinGetPos($hWnd)  
If @error Then  
    Exit  
EndIf  
  
; Move the window slightly up.  
Local $iNewX = $aPos[0]  
Local $iNewY = $aPos[1] - 10  
  
; Reposition the window (you can leave the width/height unchanged).  
WinMove($winTitle, "", $iNewX, $iNewY)  
  
; Optionally, focus the window again (to ensure it is in the foreground).  
WinActivate($hWnd)
```



## AutoIt: Return Window Title

---

```
; Get the process ID (PID) for Notepad.  
Local $sPid = ProcessExists("notepad.exe")  
  
; Check if Notepad is running.  
If $sPid = 0 Then  
    MsgBox(0, "Error", "Notepad is not running.")  
    Exit  
EndIf  
  
; Find the window handle for Notepad.  
Local $hWnd = WinGetHandle("[CLASS:Notepad]")  
  
; Check if the window handle is valid.  
If $hWnd = "" Then  
    MsgBox(0, "Error", "Unable to find Notepad window.")  
    Exit  
EndIf  
  
; Retrieve the window title.  
Local $sTitle = WinGetTitle($hWnd)  
  
; Display the window title.  
If $sTitle <> "" Then  
    MsgBox(0, "Notepad Window Title", "The title of the Notepad window is: " &  
$sTitle)  
Else  
    MsgBox(0, "Error", "Failed to retrieve the window title.")  
EndIf
```



## AutoIt: WinActivate, Take Control of Window

---

I used this to take control of a web based installation, one that normally required user intervention. I was able to perform the install without user intervention.

```
#include <AutoItConstants.au3>

; Suppress the errors.
Global $iErrorMode = 0

; Kill the InteractionWorkspace.exe process.
Run("TASKKILL /F /IM DesktopEdition.exe", "", @SW_HIDE)

; Run the Workspace Desktop Edition maintenance command.
Run("rundll32.exe dfshim.dll,ShArpMaintain Desktop Edition.application,
Culture=neutral, PublicKeyToken=0000000000000000, processorArchitecture=msil",
"", @SW_HIDE)

; Pause to allow the process to start.
Sleep(1000)

; Wait for the maintenance window to appear.
While Not WinActivate("Desktop Edition Maintenance")
    Sleep(200)
Wend

; Switch focus to the maintenance window and send TAB.
WinActivate("Desktop Edition Maintenance")
Send("+{TAB}") ; Shift + Tab

; Ensure the window is still active, then send "OK".
WinActivate("Desktop Edition Maintenance")
Send("OK")

; Exit the script.
Exit
```



## 9. Bash macOS

---

### Brief Overview

**Bash** (Bourne Again Shell) is the backbone of task automation on Unix-based systems, including macOS. First released in 1989 as part of the GNU project, Bash was created to improve upon the original Bourne Shell by introducing modern features and enhanced flexibility. Over time, it has become a cornerstone of scripting, empowering users to streamline repetitive tasks, manage processes, and interact with their operating system efficiently—all through a clean and powerful command-line interface.

On macOS, Bash scripts unlock a world of automation possibilities, enabling users to manage workflows that range from simple file operations to complex system administration. Whether it's organizing directories, scheduling backups, or configuring software environments, Bash offers a straightforward yet versatile syntax that simplifies a wide variety of tasks. Its seamless integration with Unix-based systems makes it a trusted tool for developers, IT professionals, and power users aiming to boost productivity without relying on additional software.

One of Bash's greatest strengths is its lightweight nature combined with robust capabilities for handling intricate logic, text processing, and process control. Despite newer shells like Zsh becoming the default on macOS, Bash remains a widely used and reliable tool, known for its extensive community support, broad compatibility, and cross-platform functionality. These qualities ensure it continues to be an essential part of Unix-based systems, providing an accessible and efficient scripting environment for users of all levels.

From automating routine tasks to developing sophisticated workflows, Bash remains an invaluable resource for anyone working in a Unix environment. Its enduring popularity is a testament to its power, flexibility, and ability to simplify complex tasks, making it a cornerstone of task automation across macOS and beyond.



## Why is it still practical today?

- **Universal System Integration:** Bash is natively available on most Unix-based systems, making it a universal tool for automating tasks on servers, desktops, and development environments.
- **Ease of Use:** The Bash scripting syntax is simple and intuitive, with a focus on text processing, system commands, and automation. While more complex operations can require advanced scripting techniques, Bash remains user-friendly for everyday tasks.
- **Powerful Automation:** With built-in support for piping, redirection, and conditional statements, Bash scripts can be used for complex automation workflows, including system monitoring, data processing, and task scheduling.
- **Extensive Toolset:** Bash scripts can utilize a vast array of command-line tools and utilities, making it a highly versatile scripting language for tasks such as file manipulation, networking, and process management.
- **Open Source and Community-Driven:** As part of the GNU project, Bash is open-source and supported by an active community. This means continuous updates, a wealth of online resources, and plenty of community-contributed libraries and scripts.

Bash continues to be a foundational tool for automating system tasks in Unix-like environments. Whether used for simple scripts, managing complex systems, or writing deployment pipelines, Bash remains practical, flexible, and indispensable for anyone working in Unix-based environments.

We use shell to execute our bash scripts. So, when naming your scripts, use the .sh extension.

**Examples:** Script.sh, My\_Script.sh, Copy-Script.sh



## Bash: Design

---

<https://developer.apple.com/library/archive/documentation/OpenSource/Conceptual/ShellScripting>

### Basic Construction

---

```
#!/bin/bash

# Disable command echoing.
set +v

# Set the window title (works in Terminal).
echo -ne "\033]0;Greeting Script\007"

# Clear the terminal screen.
clear

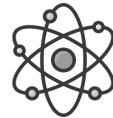
# Prompt the user for their name.
read -p "What is your name? " Name

# Check if the user entered a name or left it empty.
if [ -z "$Name" ]; then
    echo "You didn't enter a name. Goodbye!"
    exit 1
fi

# Display a personalized greeting.
echo "Hello, $Name! Have a great day!"

# Pause the script to allow the user to see the message.
read -n 1 -s -r -p "Press any key to continue..."

# Exit the script successfully.
exit 0
```



## Bash: Script Execution

---

### Line Endings

---

Understanding line endings is crucial for compatibility across different operating systems. When working with scripts, text files, or source code that will be used on multiple platforms, it's important to ensure the line endings are consistent. If you're encountering odd errors with your scripts, they fail even though the syntax appears correct, check the line endings.

A common line ending error—**unexpected end of file**.

```
# Check line endings.  
cat -A script.sh  
  
# Fix line endings.  
sed -i '' 's/\r//' script.sh
```

### Make Script Executable

---

```
chmod +x script.sh
```

### Remove Executable Permission

---

```
chmod -x script.sh
```



## Bash: User Operations

---

### Return Logged in User

---

```
#!/bin/bash

# Try to get the logged-in user using ioreg.
loggedInUser=$(ioreg -n Root -d1 -a | xpath
'plist/dict/key[.="IOConsoleUsers"]/following-sibling::array/dict/key[.="kCGSSessionOnConsoleKey"]/following-sibling::*[1][name()="true"]/../key[.="kCGSSessionUserNameKey"]/following-sibling::string[1]/text()' 2>/dev/null)

# If no user is found, try using scutil to get the console user.
if [ -z "$loggedInUser" ]; then
    loggedInUser=$((/usr/sbin/scutil <<< "show State:/Users/ConsoleUser" | awk
'/Name :/ && ! /loginwindow/ {print $3}')
fi

# Output the logged-in user.
echo "The username is $loggedInUser"
```

### Create New Account

---

```
#!/bin/bash

# Get the highest UID in use and calculate the next UID.
highestUID=$((/usr/bin/dscl . -list /Users UniqueID | /usr/bin/awk 'BEGIN {m=0}
$2>m {m=$2} END {print m}')
if [[ -z "$highestUID" || ! "$highestUID" =~ ^[0-9]+$ ]]; then
    echo "Error: Unable to determine the highest UID."
    exit 1
fi

nextUID=$((highestUID + 1))

# Create the user home directory if it doesn't already exist.
userHome="/Users/MacAdmin"
if [[ ! -d "$userHome" ]]; then
    mkdir -p "$userHome"
```



```
if [[ $? -ne 0 ]]; then
    echo "Error: Failed to create directory $userHome."
    exit 1
fi
else
    echo "Directory $userHome already exists."
fi

# Create the user account.
/usr/bin/dscl . create "$userHome"
if [[ $? -ne 0 ]]; then
    echo "Error: Failed to create user account."
    exit 1
fi

/usr/bin/dscl . create "$userHome" UserShell /bin/zsh
/usr/bin/dscl . create "$userHome" RealName "Mac Admin"
/usr/bin/dscl . create "$userHome" UniqueID "$nextUID"
/usr/bin/dscl . create "$userHome" PrimaryGroupID 80 # Use 80 for admin group

# Confirm account creation.
if [[ $? -eq 0 ]]; then
    echo "User 'MacAdmin' successfully created with UID $nextUID."
else
    echo "Error: User creation failed."
    exit 1
fi

echo "Press any key to continue..."
read -n 1 -s
```

## Delete Existing Account

---

```
#!/bin/bash

# Function to handle errors.
handle_error() {
    echo "Error: $1"
    exit 1
}
```



```
# Set the username variable.
USER="MacAdmin"

# Ensure the user account is not hidden.
echo "Ensuring user $USER is not hidden..."
/usr/bin/dscl . create "/Users/$USER" IsHidden 0 || handle_error "Failed to
unhide user account."

# Remove any associated share points.
echo "Removing share point for $USER..."
/usr/bin/dscl . delete "/SharePoints/$USER" 2>/dev/null || echo "No share point
found for $USER."

# Remove the user's home directory.
if [ -d "/Users/$USER" ]; then
    echo "Deleting home directory for $USER..."
    rm -rf "/Users/$USER" || handle_error "Failed to delete home directory."
else
    echo "No home directory found for $USER."
fi

# Delete the user account.
echo "Deleting user account $USER..."
/usr/bin/dscl . delete "/Users/$USER" || handle_error "Failed to delete user
account."

# Final Confirmation
echo "User $USER has been successfully removed.

echo "Press any key to exit..."
read -n 1 -s
```

## Create Account, Set as Admin, and Show User

---

```
#!/bin/bash

# Return Next ID
highestUID=$( dscl . -list /Users UniqueID | /usr/bin/awk '$2>m {m=$2}' END {
print m } )
nextUID=$(( highestUID+1 ))

# Make Folder
mkdir /Users/MacAdmin
```



```
# Create Account
/usr/bin/dscl . create "/Users/MacAdmin"
/usr/bin/dscl . create "/Users/MacAdmin" UserShell /bin/zsh
/usr/bin/dscl . create "/Users/MacAdmin" RealName "Mac Admin"
/usr/bin/dscl . create "/Users/MacAdmin" UniqueID "$nextUID"
/usr/bin/dscl . create "/Users/MacAdmin" PrimaryGroupID 20

# Add Secure Token
# Ref: https://support.apple.com/en-gu/guide/deployment/dep24dbdcf9e/web
/usr/bin/dscl . append "/Users/MacAdmin" AuthenticationAuthority
";DisabledTags;SecureToken"

# Make Admin
/usr/bin/dscl . append "/Groups/admin" GroupMembership "MacAdmin"

# Set Password - note the escaped character
/usr/bin/dscl . passwd "/Users/MacAdmin" "Password_Here"

# Set Home Directory
/usr/bin/dscl . create "/Users/MacAdmin" NFSHomeDirectory "/Users/MacAdmin"

# Not Hidden
/usr/bin/dscl . create "/Users/MacAdmin" IsHidden 0

# Add Secure Token
# Ref: https://support.apple.com/en-gu/guide/deployment/dep24dbdcf9e/web
/usr/bin/dscl . append "/Users/MacAdmin" AuthenticationAuthority
";DisabledTags;SecureToken"

echo "Press any key to continue..."; read -n 1 -s
exit 0
```



## Bash: File & Folder Operations

---

### Copy File

---

```
#!/bin/bash
# Be aware that copying and pasting code can cause line issues. See: Link

cp '/Volumes/SomeMountedDMG/setup.pkg' '/Applications/setup.pkg'

echo "Press any key to continue..."

read -n 1 -s
```

### Delete File

---

```
#!/bin/bash
# Be aware that copying and pasting code can cause line issues. See: Link

rm '/Applications/setup.dmg'

echo "Press any key to continue..."; read -n 1 -s
```

### Copy Folder

---

```
#!/bin/bash
# Be aware that copying and pasting code can cause line issues. See: Link

# recursive
cp -r '/Volumes/SomeMountedDMG' '/Applications'
# force
cp -rf '/Volumes/SomeMountedDMG' '/Applications'

echo "Press any key to continue..."; read -n 1 -s
```



## Delete Folder

---

```
#!/bin/bash
# Be aware that copying and pasting code can cause line issues. See: Link

# recursive
sudo rm -r '/Applications/setup.app'
# force
rm -rf '/Applications/setup.app'

echo "Press any key to continue..."; read -n 1 -s
```

## Change File/Folder Permissions

---

```
#!/bin/bash

# Read/Write Everyone
sudo chmod a+rwx './test-folder'
sudo chmod -R 777 '/Library/Internet Plug-Ins/JavaAppletPlugin.plugin'

echo "Press any key to continue..."; read -n 1 -s
```

## Mount & Attach DMG

---

```
#!/bin/bash

hdiutil attach '/Applications/Setup.dmg'

# Silent
hdiutil attach -nobrowse '/Applications/Setup.dmg'

hdiutil mount '/Applications/Setup.dmg'

echo "Press any key to continue..."; read -n 1 -s
```



## Detach & Unmount DMG

---

```
#!/bin/bash

hdiutil detach '/Applications/Citrix Workspace'
hdiutil unmount '/Volumes/Citrix Workspace'

echo "Press any key to continue..."; read -n 1 -s
```

## Clear Intune Company Portal Cache

---

```
#!/bin/bash

sudo rm -rf ~/Library/Caches/com.microsoft.CompanyPortalMac
sudo rm -rf ~/Library/Containers/com.microsoft.CompanyPortalMac
sudo rm -rf ~/Library/Preferences/com.microsoft.CompanyPortalMac.plist
sudo rm -rf "~/Library/Application Support/com.microsoft.CompanyPortalMac"
sudo rm -rf "~/Library/Application Scripts/com.microsoft.CompanyPortalMac"
killall cfprefsd

echo "Press any key to continue..."; read -n 1 -s
```



## Bash: Process Operations

---

### Open .App

---

```
#!/bin/bash
open -a '/Applications/Firefox.app'

# Wait
open -g -W '/Applications/Firefox.app'
echo "Press any key to continue..."; read -n 1 -s
```

### Kill Process

---

```
#!/bin/bash

sudo pkill -9 'Google Chrome' # -9 is force kill
sudo pkill -9 'firefox'

echo "Press any key to continue..."; read -n 1 -s
```



## Bash: OS Operations

---

### Set Time & Date

---

```
#!/bin/bash

current_datetime=$(date +"%Y%m%d_%H%M%S")
echo "Current date and time: $current_datetime"
echo "Press any key to continue..."; read -n 1 -s
```

### Set Time Zone

---

```
#!/bin/bash

sudo systemsetup -listtimezones
sudo systemsetup -settimezone timezone
sudo systemsetup -gettimezone

echo "Press any key to continue..."; read -n 1 -s
```

### Unload Service

---

```
#!/bin/bash

launchctl unload '/Library/LaunchDaemons/com.oracle.java.Helper-Tool.plist'
2>/dev/null

echo "Press any key to continue..."; read -n 1 -s
```



## Bash: Miscellaneous

---

### Create Log File

---

```
#!/bin/bash
LOG_FILE="/var/log/app_uninstall.log"

log() {
    echo "$(date '+%Y-%m-%d %H:%M:%S') - $1" | tee -a "$LOG_FILE"
}

log "Starting App Uninstall Process..."
```

### Fix Sudoers Permissions Error

---

```
osascript -e 'do shell script "chmod 0440 /etc/sudoers;chown root:wheel /etc/sudoers" with administrator privileges'
```

### Sign PKG with Certificate

---

```
#!/bin/bash

sudo productsign --sign "Developer ID Installer: Eddie Jackson (XV9Q4X5000)" \
PKG-unsigned.pkg PKG-signed.pkg

echo "Press any key to continue..."; read -n 1 -s
```

#### | output |

```
productsign: using timestamp authority for signature
productsign: signing product with identity "Developer ID Installer: Eddie
Jackson (XV9Q4X5000)" from keychain
/Users/eddiejackson/Library/Keychains/login.keychain-db
productsign: adding certificate "Developer ID Certification Authority"
productsign: adding certificate "Apple Root CA"
productsign: Wrote signed product archive to PKG-signed.pkg
```



## Check Signature on App

---

```
#!/bin/bash

sudo pkgutil --check-signature setup.app

echo "Press any key to continue..."; read -n 1 -s
```

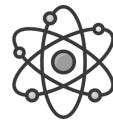
### | output |

#### Good

```
Package "Setup.app":
  Status: signed by a certificate trusted by macOS
  Certificate Chain:
    1. Developer ID Application: Citrix Systems, Inc. (S272Y5R111)
      Expires: 2027-02-01 22:12:15 +0000
      SHA256 Fingerprint:
        C8 C0 8D 1A C8 B8 45 48 15 80 D1 CF 9D 1E F5 E4 B8 F8 D5 67 E5 16
        D1 52 F9 5C 72 8E 68 00 00 00
    -----
    2. Developer ID Certification Authority
      Expires: 2027-02-01 22:12:15 +0000
      SHA256 Fingerprint:
        7A FC 9D 01 A6 00 03 A2 DE 96 37 93 6D 4A FE 68 09 0D 2D E1 00 03
        F2 9C 88 CF B0 B1 BA 00 00 00
    -----
    3. Apple Root CA
      Expires: 2035-02-09 21:40:36 +0000
      SHA256 Fingerprint:
        B0 B1 73 0E CB C7 FF 45 05 14 2C 49 F1 29 00 6E DA 6B 00 ED 7E 2C
        68 C5 BE 00 B5 A1 10 00 00 00
```

#### Bad

```
Package "Setup.app":
  Status: package is invalid (checksum did not verify)
```



## Install PKG Using Installer

---

```
#!/bin/bash

cd /Applications
installer -pkg '/Volumes/Citrix Workspace/Install Citrix Workspace.pkg' -target /
echo "Press any key to continue..."; read -n 1 -s
```

## Install Citrix Client from DMG

---

When combined with a packaging tool like Packages (<http://s.sudre.free.fr/>), this script enables the creation of custom PKG files that can be deployed through desktop management systems (SCCM, Intune, Ivanti). Upon deployment or execution, the script will automate the installation process seamlessly.

```
#!/bin/bash

cd /Applications

hdiutil attach -nobrowse '/Applications/Citrix.dmg'; sleep 3

sudo ditto '/Volumes/Citrix Workspace/Uninstall Citrix Workspace.app'
'/Applications/Uninstall Citrix Workspace.app'; sleep 3

installer -pkg '/Volumes/Citrix Workspace/Install Citrix Workspace.pkg' -target /
; sleep 3

hdiutil detach '/Volumes/Citrix Workspace'; sleep 3

rm '/Applications/Citrix.dmg'

echo "Done!"; sleep 10
```



## Bash: Logic Examples

---

### IF Condition, Unblock Website

---

```
#!/bin/bash

# Check if the website entry exists
if grep -q "www.tiktok.com" "/private/etc/hosts"; then
    echo "Removing website entry..."
    sudo sed -i '' '/www.tiktok.com/d' "/private/etc/hosts"
fi
cat /private/etc/hosts
echo "Press any key to continue..."; read -n 1 -s
```

### IF NOT Condition, Block Website

---

```
#!/bin/bash

# Check if the website entry exists
if ! grep -q "www.tiktok.com" "/private/etc/hosts"; then
    echo "Adding website entry..."
    echo '127.0.0.1 www.tiktok.com' |
        sudo tee -a "/private/etc/hosts" > /dev/null
fi
cat /private/etc/hosts
echo "Press any key to continue..."; read -n 1 -s
```



## Logic Script

---

This Bash script evaluates different logical operators (AND, OR, NOT, and XOR) using boolean variables to determine a person's eligibility to enter a club based on whether they have a ticket or a membership. It first checks if both conditions are true (AND), then if at least one is true (OR), then negates the ticket condition (NOT), and finally checks if exactly one condition is true (XOR). The script displays the results of these logical evaluations.

```
#!/bin/bash

# Imagine we are checking if someone is eligible to enter a club.
has_ticket=true # The person has a ticket.
is_member=false # The person is not a member.

# AND Operator: Both conditions must be true
# A person can enter the club only if they have a ticket AND are a member.
if [[ $has_ticket == true && $is_member == true ]]; then
    echo "Can enter with both ticket and membership? true"
else
    echo "Can enter with both ticket and membership? false"
fi
# Output: false

# OR Operator: Only one condition needs to be true.
# A person can enter the club if they have a ticket OR are a member.
if [[ $has_ticket == true || $is_member == true ]]; then
    echo "Can enter with either a ticket or membership? true"
else
    echo "Can enter with either a ticket or membership? false"
fi
# Output: true

# NOT Operator: Reverses the condition.
# We want to check if a person doesn't have a ticket.
if [[ ! $has_ticket == true ]]; then
    echo "Does not have a ticket? true"
else
    echo "Does not have a ticket? false"
fi
# Output: false

# XOR Operator: True only if exactly one condition is true, but not both.
```



```
# A person can enter the club if they either have a ticket OR are a
# member, but not both.
if [[ $has_ticket != $is_member ]]; then
    echo "Can enter with either a ticket or membership, but not both? true"
else
    echo "Can enter with either a ticket or membership, but not both? false"
fi
# Output: true
```



## 10. Scripted Management: A.K.A. Automation

---

### Introduction to Automation

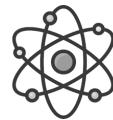
By this point, you've got a solid grasp of the scripting basics, so I hope the concepts we use here aren't too overwhelming. Before diving into the topics in C# programming, I want to walk you through some practical examples of how to use code to automate the installation of software (this could easily be extended to app uninstalls and other desktop management tasks). Whether you're using Batch or PowerShell (or Python or AutoIt language, etc.), scripted installs can save you a lot of time by streamlining the setup process and ensuring that software installs correctly, with all necessary settings and configurations in place, each and every time the scripted task is executed.

One of the greatest strengths of automation is its ability to manage tasks that would otherwise demand manual effort—often involving numerous setup steps and extensive documentation. By automating these processes, you can minimize the reliance on lengthy, often neglected documentation (you'd be surprised how shockingly outdated documentation is in many companies), ensure consistency in the execution every time, and significantly reduce errors that commonly occur during manual operations.

In this section, we'll start with some straightforward installation scripts in both Batch and PowerShell, gradually working our way up to more complex setups. I'll provide three examples for each scripting language, each progressively more involved, culminating in the final example that handles multiple tasks—installs, importing certificates, and editing the registry—while showcasing the power of scripted automation.

The examples in this section will serve as a perfect segue into the following key areas:

- 15. Integration** – Bringing together everything you have learned.
  - 16. Basic Flowcharting** – Mapping code processes visually.
  - 17. Compilation & Packaging** – Converting scripts into executables.
  - 18. Deployment** – Distributing solutions across systems.
-



## Batch Script Examples

---

### Example 1: Basic Software Installation

The first example will cover a very simple, typical software installation using MSIEEXEC in a Batch script. This is one of the most common methods for installing MSI-based applications. The MSI could easily be substituted for an EXE with silent parameters.

```
@ECHO OFF
:: Simple MSI Installation. Note the parameters we used.
MSIEXEC /I "C:\Batch\setup.msi" /QN /NORESTART
ECHO Done!
PAUSE & EXIT /B 0
```

**Explanation—**This script silently installs the MSI file located at C:\Batch\setup.msi using the /QN option, meaning there will be no user interaction during the install. The /NORESTART option ensures the machine doesn't automatically restart after the installation. Mind the PAUSE.

### Example 2: Custom Installation & Create a Shortcut

This example adds a bit more complexity by creating a shortcut to the installed application in the C:\Batch folder, along with the basic software installation.

```
@ECHO OFF
:: Install the software. Note the silent parameters.
MSIEXEC /I "C:\Batch\setup.msi" /QN /NORESTART

:: Create a shortcut in the C:\Batch folder.
SET TARGET="C:\Batch\App.exe"
SET SHORTCUT="C:\Batch\App.lnk"
ECHO Creating shortcut...
PowerShell -Command "$WScript = New-Object -ComObject
WScript.Shell; $Shortcut = $WScript.CreateShortcut('%SHORTCUT%');
$Shortcut.TargetPath = '%TARGET%'; $Shortcut.Save()"
ECHO Done! & PAUSE & EXIT /B 0
```

**Explanation—**After installing the software, the script creates a shortcut in C:\Batch to App.exe. We're using PowerShell to create the shortcut, but the rest of the process remains within the Batch environment. The use of TARGET and SHORTCUT variables makes this script flexible. Here's a challenge: How to add a shortcut to the user's desktop from the System account?



### Example 3: Installing Software, Importing a Certificate & Editing Registry

This example is more complex, including software installation, a certificate import, and registry modifications in both HKLM and HKU. Be sure to run as administrator, as this script writes to HKLM & uses CertUtil for a Root cert, which requires elevated rights.

```
@ECHO OFF
SETLOCAL EnableDelayedExpansion
:: Set current directory as our default working directory.
:: This is better than defining static paths.
CD %~dp0
:: Install the software silently.
MSIEXEC /I "setup.msi" /QN /NORESTART

:: Import a certificate into the user's certificate store.
ECHO Importing certificate... & TIMEOUT /T 2 >nul
CERTUTIL -ADDSTORE "Root" "App.cer"
ECHO.

:: Add registry entries for the application.
ECHO Adding registry keys... & TIMEOUT /T 2 >nul
ECHO Adding hklm key HKEY_LOCAL_MACHINE\Software\TEST...&TIMEOUT /T 2 >nul
REG ADD "HKEY_LOCAL_MACHINE\Software\TEST" /V "myKey" /D 1 /F /REG:64 >nul

:: Return User SID
FOR /F "tokens=3" %%A IN ('REG QUERY
"HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Authentication\LogonUI" /V
"SelectedUserSID" /REG:64 2>nul ^| FIND "REG_SZ"') DO (
    SET "uSID=%~A"

ECHO Adding hku key HKEY_USERS!\uSID!\Software\TEST...&TIMEOUT /T 2>nul
REG ADD "HKEY_USERS!\uSID!\Software\TEST" /V "myKey" /D 1 /F /REG:64 >nul
)

ENDLOCAL & ECHO. & ECHO Done! & ECHO. & PAUSE
EXIT /B 0
```

**Explanation—**After the software is installed, the script imports a certificate into the user's machine using CERTUTIL. It then adds registry keys both to the HKLM hive (for machine-wide settings) and the HKU hive (for user-specific settings), where !uSID! is a dynamic variable for the primary user SID returned from HKU. Here's a challenge: How to create a scheduled task from the System account to repair the setup.msi on-demand?



## PowerShell Script Examples

---

### Example 1: Basic Software Installation

The following PowerShell script uses MSIEXEC for a silent installation of an MSI file.

```
# Simple MSI Installation. Note the parameters used.  
Start-Process MSIEEXEC -ArgumentList '/I', 'C:\PowerShell\setup.msi',  
'/QN', '/NORESTART' -NoNewWindow -Wait  
  
Write-Host "Done!"  
  
pause
```

### Explanation

#### **Start-Process**

This cmdlet is used to start a new process—in this case, msieexec.exe.

#### **-ArgumentList**

This is where we pass the parameters to msieexec. The /I option specifies that we want to install the MSI file located at C:\PowerShell\setup.msi. The /QN option ensures a silent installation without user interaction, and /NORESTART prevents the machine from restarting automatically after the installation.

#### **-NoNewWindow**

This flag ensures that the process runs in the same window (no new console window is opened).

#### **-Wait**

This ensures that the script waits for the msieexec process to complete before moving on to the next command.



## Example 2: Custom Installation & Create a Shortcut

In this example, we add complexity by not only installing the software silently but also creating a shortcut to the installed application in the C:\PowerShell folder using PowerShell.

```
# Install the software silently with msieexec.  
Start-Process MSIEEXEC -ArgumentList '/I', "C:\PowerShell\setup.msi",  
'/QN', '/NORESTART' -NoNewWindow -Wait  
  
# Define the target application and shortcut paths.  
$Target = "C:\PowerShell\App.exe"  
$Shortcut = "C:\PowerShell\App.lnk"  
  
Write-Host "Creating shortcut..."  
  
# Create a shortcut using WScript.Shell COM object.  
$WScript = New-Object -ComObject WScript.Shell  
$ShortcutObj = $WScript.CreateShortcut($Shortcut)  
$ShortcutObj.TargetPath = $Target  
$ShortcutObj.Save()  
  
Write-Host "Done!"  
pause
```

## Explanation

### Install the Software Silently

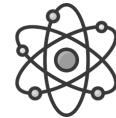
The script uses Start-Process with MSIEXEC.exe to silently install the application. The /I option points to the setup.msi file, /QN suppresses user interaction, and /NORESTART prevents automatic system restarts.

### Defining Target and Shortcut

We define the paths for the target application (App.exe) and the location where the shortcut (App.lnk) will be created. These paths can be adjusted as needed for flexibility.

### Creating the Shortcut

We use the WScript.Shell COM object in PowerShell to create the shortcut. This is done by first creating a shortcut object with \$WScript.CreateShortcut(\$Shortcut) and then assigning the TargetPath to point to the application (App.exe). Finally, \$ShortcutObj.Save() saves the shortcut to the specified location.



### Example 3: Installing Software, Importing a Certificate & Editing Registry

This example is more complex, including software installation, a certificate import, and registry modifications in both HKLM and HKU. Be sure to run as administrator, as this script writes to HKLM and imports a cert into Root, which requires elevated rights.

```
# Enable strict error handling.
$ErrorActionPreference = 'SilentlyContinue' # stop

# Set current directory as our default directory.
# This is better than using static paths.
Set-Location -Path $PSScriptRoot

# Install the software
Write-Host "Installing software..."
Start-Process MSIEEXEC -ArgumentList '/I', "$PSScriptRoot\setup.msi",
'/QN', '/NORESTART' -NoNewWindow -Wait
Write-Host "Software installed.`n"

# Import a certificate into the user's certificate store.
Write-Host "Importing certificate..."
Start-Sleep -Seconds 2
$CertificatePath = "C:\PowerShell\App.cer"
$StoreName = "Root"
$StoreLocation = "LocalMachine"

if (Test-Path $CertificatePath) {
    $Certificate = New-Object
    System.Security.Cryptography.X509Certificates.X509Certificate2
    $Certificate.Import($CertificatePath)

    $Store = New-Object
    System.Security.Cryptography.X509Certificates.X509Store($StoreName,
    $StoreLocation)
    $Store.Open("MaxAllowed")
    $Store.Add($Certificate)
    $Store.Close()

    Write-Host "Certificate imported successfully into the $StoreName
    store."
} else {
    Write-Warning "Certificate file not found at $CertificatePath."
}

# Add registry entries for the application.
```



```
Write-Host "Adding registry keys..."  
Start-Sleep -Seconds 2  
  
# Add to HKEY_LOCAL_MACHINE.  
$HKLMPath = "HKLM:\Software\TEST"  
if (-not (Test-Path $HKLMPath)) {  
    New-Item -Path $HKLMPath -Force | Out-Null  
}  
  
Set-ItemProperty -Path $HKLMPath -Name "myKey" -Value 1 -Force  
  
Write-Host "Added registry key to HKEY_LOCAL_MACHINE: $HKLMPath.\`n"  
  
# Retrieve the User SID from the registry.  
$LogonUIPath =  
"HKLM:\SOFTWARE\Microsoft\Windows\CurrentVersion\Authentication\LogonUI"  
$UserID = (Get-ItemProperty -Path $LogonUIPath -Name 'SelectedUserSID'  
-ErrorAction 'SilentlyContinue').SelectedUserSID  
  
if ($UserID) {  
    Write-Host "User SID: $UserID"  
  
    # Define the target path in HKEY_USERS.  
    $RegistryPath = "Registry::HKEY_USERS\$UserID\Software\TEST"  
  
    # Create the registry key if it doesn't exist.  
    if (-not (Test-Path $RegistryPath)) {  
        Write-Host "Creating registry path: $RegistryPath"  
        New-Item -Path $RegistryPath -Force | Out-Null  
    }  
  
    # Set the registry value  
    Set-ItemProperty -Path $RegistryPath -Name "myKey" -Value 1 -Force  
    Write-Host "Registry key added successfully: $RegistryPath"  
} else {  
    Write-Warning "User SID not found. Ensure the SelectedUserSID exists  
in the LogonUI registry key."  
}  
  
Write-Host "`nDone!"  
pause
```



## Explanation

### Silent Software Installation

The script uses Start-Process to invoke MSIEXEC.exe for a silent installation. The /I option specifies the path to our setup.msi file, while /QN suppresses user interaction, ensuring the process runs silently. The /NORESTART option prevents the system from automatically restarting after installation.

### Importing a Certificate

The script checks the presence of the App.cer file before importing it into the "Root" certificate store using PowerShell's built-in methods. It creates an X509Certificate2 object to load the certificate and adds it to the "Root" store under the LocalMachine location. This process ensures the system trusts the certificate, which may be necessary for the secure operation of the application.

### Modifying Registry Entries

The script creates and updates a registry key under HKEY\_LOCAL\_MACHINE\Software\TEST, adding a value for configuration. It then dynamically retrieves the currently logged-in user's SID from the SelectedUserId value in the LogonUI registry key. Using this SID, the script creates or updates a registry key under HKEY\_USERS for the active user, facilitating user-specific configurations.

### Dynamic User Handling

By programmatically identifying the active user's SID, the script ensures that any registry changes under HKEY\_USERS are applied to the correct user profile. This eliminates the need to hardcode user-specific paths, enhancing the script's flexibility and scalability across different environments.

### Further Exercises

- Add Logging to Track Script Execution
- Validate Certificate Integrity Before Importing
- Check for Existing Software Installation and Skip if Present
- Handle Errors Gracefully with Try-Catch Blocks
- Prompt User for Installation Parameters (Custom Paths, Options)
- Automate Software Uninstallation if Already Installed
- Set Custom Permissions for Installed Files and Registry Keys



What else can you do with automation? Pretty much anything you can think of. Software install and uninstalls are common, but here are some other ideas.

### Things for you to try:

- **Managing User Profiles:** Configure default settings, clear temporary files, or apply user-specific policies.
- **Updating System Configurations:** Modify registry settings, adjust environment variables, or update configuration files.
- **Managing Local Groups and Permissions:** Add or remove users from groups, such as the local Administrators group, to enforce security policies.
- **Monitoring and Reporting:** Generate logs or alerts for system performance, disk usage, or failed login attempts.
- **System Cleanup and Optimization:** Remove unnecessary files, defragment drives, or clear system caches.
- **Applying Security Updates:** Automate patch installations, disable unnecessary services, or enforce password policies.
- **Customizing the User Interface:** Create desktop shortcuts, pin apps to the taskbar, or configure wallpaper and themes.
- **Provisioning New Devices:** Set up new workstations with preconfigured software, settings, and security baselines.

If you can think it, you can bring it into existence. Feel the power yet?

Be sure to check out the *Unified Solutions* section.



# PROGRAMMING FUNDAMENTALS



## 11. Foundational

---

Object-Oriented Programming (OOP) introduces powerful principles like classes, properties, methods, inheritance, encapsulation, polymorphism, and abstraction. These concepts aren't just technical jargon—they're the building blocks that make modern programming click. By structuring code around real-world objects and their behaviors, OOP keeps things organized, reduces repetition, and makes maintaining software much less of a headache. Whether you're creating a small app or tackling a massive system, OOP has your back.

What really sets OOP apart is how it handles complexity. Features like controlled data access and abstraction simplify things by showing you only what you need while hiding the messy details. Inheritance lets you reuse and build on existing code effortlessly, and polymorphism makes it easy to work with different data types or behaviors under one consistent approach. These tools aren't just for show—they're what make OOP ideal for building software that's flexible, scalable, and ready to evolve.

OOP isn't just about writing code; it's about writing better code, especially when working with a team. Clear, modular structures help everyone stay on the same page, make testing easier, and speed up development. From boosting security with encapsulation to solving complex challenges efficiently, OOP equips developers to create innovative, reliable software that meets today's high demands—and tomorrow's challenges, too.

The content that follows is not intended to be an exhaustive compendium of programming knowledge—that would require volumes (see Donald Knuth's *The Art of Computer Programming* series). Instead, this section offers a concise overview of the essentials.

Let's dive into these core concepts.



## Classes

---

A class is a blueprint for creating objects (instances) in object-oriented programming (OOP). It defines the properties and behaviors (methods) that objects of that class will have.

- **Properties** represent the data or state of an object. They are variables associated with the class.
- **Methods** represent the actions or behaviors that objects of that class can perform. They are functions associated with the class.

Classes provide a way to organize and encapsulate related data and functionality.

## Properties

---

Properties, also known as attributes or fields, represent the state or data associated with an object. They define the characteristics or features of an object.

- Properties can have different data types, such as integers, strings, arrays, or even other objects.
- Each object created from a class has its own set of property values.

Properties hold the essential data for each object, allowing it to maintain its unique state while enabling the class to define and manage its characteristics.



## Methods

---

Methods are functions defined within a class that define the behavior of objects instantiated from that class. They encapsulate the operations or actions that objects can perform.

- Methods can access and modify the properties of the object they belong to.
- They can also interact with other objects or call other methods.

Methods provide the functionality that drives the behavior of an object, allowing it to interact with its own properties, other objects, and the wider system.

## Inheritance

---

Inheritance is a fundamental concept in OOP that allows a class to inherit properties and methods from another class. The class that inherits is called the subclass or derived class, and the class being inherited from is called the superclass or base class.

- **Single Inheritance:** A subclass inherits from one superclass.
- **Multiple Inheritance:** A subclass inherits from more than one superclass (not supported in all programming languages).
- **Hierarchical Inheritance:** Multiple subclasses inherit from a single superclass.
- **Multilevel Inheritance:** A subclass inherits from another subclass, forming a chain.

Inheritance promotes code reusability and establishes a natural hierarchy between classes.



## Encapsulation

---

Encapsulation is the concept of bundling the data (properties) and methods that operate on the data into a single unit, or class. It restricts direct access to some of the object's components, which is a means of preventing accidental interference and misuse of the data.

- **Private:** Members are accessible only within the same class.
- **Protected:** Members are accessible within the same class and by derived class instances.
- **Public:** Members are accessible from any other code.

Encapsulation helps in maintaining the integrity of the data by providing controlled access through methods (getters and setters).

## Polymorphism

---

Polymorphism allows objects of different classes to be treated as objects of a common superclass. It is the ability of different objects to respond to the same method call in different ways.

- **Compile-time Polymorphism (Method Overloading):** Multiple methods with the same name but different parameters.
- **Runtime Polymorphism (Method Overriding):** A subclass provides a specific implementation of a method that is already defined in its superclass.

Polymorphism enhances flexibility and integration of code.



## Abstraction

---

Abstraction is the concept of hiding the complex implementation details and showing only the essential features of the object. It helps in reducing programming complexity and effort.

- **Abstract Class:** A class that cannot be instantiated and is designed to be subclassed. It can have abstract methods (without implementation) and concrete methods (with implementation).
- **Interface:** A contract that defines a set of methods that a class must implement. Interfaces provide a way to achieve abstraction and multiple inheritance.

Abstraction focuses on what an object does instead of how it does it.

## Constructors and Destructors

---

- **Constructor:** A special method that is called when an object is instantiated. It initializes the object's properties.
- **Destructor:** A special method that is called when an object is destroyed. It is used to perform cleanup operations.

Constructors and destructors ensure that an object is properly initialized and cleaned up, which are ultimately used to control the behavior in our programs.



## Static Members

---

Static members belong to the class rather than any object instance. They are shared among all instances of the class.

- **Static Properties:** Variables that are shared by all instances of a class.
- **Static Methods:** Methods that can be called on the class itself, rather than on instances of the class.

Static members are useful for defining constants and utility functions.

## Access Modifiers

---

Access modifiers define the scope and visibility of class members.

- **Private:** Accessible only within the same class.
- **Protected:** Accessible within the same class and by derived classes.
- **Public:** Accessible from any other code.
- **Internal** (in some languages): Accessible within the same assembly.

Access modifiers help in implementing encapsulation and controlling access to the class members.



## Example Code

---

The following C# code demonstrates the programming principles discussed thus far.

Comments are included within the code to explain each concept—more about C# in [Section 12](#).

```
using System;
using System.Collections.Generic;
using System.Threading; // For Thread.Sleep

// Abstract class representing a general Car
// This class provides a blueprint for specific types of cars.
abstract class Car
{
    // Protected field to store the name of the car model.
    protected string _model;

    // Constructor to initialize the name of the car model.
    public Car(string model)
    {
        _model = model;
    }

    // Public property to expose the model name.
    public string Model => _model;

    // Abstract method that must be implemented by derived classes.
    public abstract string StartEngine();
}

// SportsCar class inheriting from the Car class.
class SportsCar : Car
{
    // Constructor that passes the model name to the base class constructor.
    public SportsCar(string model) : base(model) { }

    // Overriding the StartEngine method to define how a SportsCar starts.
    public override string StartEngine()
```



```
{  
    return $"{_model} roars to life with a powerful engine sound!";  
}  
}  
  
// ElectricCar class inheriting from the Car class.  
class ElectricCar : Car  
{  
    // Constructor that passes the model name to the base class constructor.  
    public ElectricCar(string model) : base(model) {}  
  
    // Overriding the StartEngine method to define how an ElectricCar starts.  
    public override string StartEngine()  
    {  
        return $"{_model} starts silently with a hum.";  
    }  
}  
  
// Main Program Entry Point  
class Program  
{  
    static void Main()  
    {  
        // Introduction to the concept.  
        Console.WriteLine("Welcome to the Car Engine Demo!");  
        Console.WriteLine("We will demonstrate polymorphism by showing how  
different types of cars start.");  
        Console.WriteLine();  
        Thread.Sleep(3000);  
  
        // Creating a list to store Car objects.  
        // This demonstrates polymorphism, as both SportsCar and ElectricCar  
        // are treated as Cars.  
        List<Car> cars = new List<Car>()  
        {  
            // Creating a SportsCar instance with the model name "Ferrari F8".  
            new SportsCar("Ferrari F8"),  
            // Creating an ElectricCar instance with the model name  
            // "Tesla Model S".  
            new ElectricCar("Tesla Model S")  
        };  
  
        // Iterating through the list of cars and calling their  
        // StartEngine method.  
        // The specific implementation of StartEngine is determined by the  
        // actual type of the object (SportsCar or ElectricCar).  
        foreach (var car in cars)
```



```
{  
    // Explain what is happening.  
    Console.WriteLine($"Starting the {car.GetType().Name}  
    ({car.Model}):");  
    Thread.Sleep(1500);  
  
    // Show the result of starting the engine.  
    Console.WriteLine(car.StartEngine());  
    Console.WriteLine();  
    Thread.Sleep(3000);  
}  
  
}  
}  
  
}
```



## 12. C# Basics

---

### Brief Overview

**C#** (pronounced "C-Sharp"), was first unveiled in 2000 and emerged as Microsoft's flagship programming language for the .NET framework. Developed under the leadership of Anders Hejlsberg, C# was designed to blend the robustness and performance of C++ with the simplicity and readability of Java. The result was a versatile, modern, and developer-friendly toolset that quickly became essential for building a wide range of applications. With its seamless integration into the .NET ecosystem, C# established itself as a cornerstone for creating Windows applications, web services, enterprise software, and beyond.

What makes C# so indispensable? Its remarkable versatility. Whether you're developing desktop support tools, standalone desktop applications, mobile apps, cloud-based services, or even game engines, C# excels in delivering efficient and scalable solutions. The language strikes a perfect balance between simplicity and power, boasting features like strong typing, object-oriented principles, and garbage collection for effective memory management.

C# extends its appeal through advanced features like LINQ (Language Integrated Query), asynchronous programming for improved performance, and dynamic types that allow greater flexibility. These innovations enable developers to write clean, efficient, and expressive code that transforms programming into an enjoyable creative process. Tasks that once required verbose boilerplate code are now streamlined and intuitive, making C# a favorite among both beginners and seasoned professionals.

Today, C# continues to thrive as a mainstay in the development world. Whether you're building cutting-edge AI solutions, creating enterprise-grade APIs, or designing the next gaming sensation with Unity, C# stands out as a reliable, versatile, and forward-thinking language that empowers developers to bring their ideas to life.



## Why is it practical today?

- **Cross-Platform Support:** With .NET 5+ and .NET Core, C# applications run seamlessly across Windows, macOS, and Linux.
- **Wide Application Scope:** C# powers everything from Windows services to web apps, mobile apps (via Xamarin or .NET MAUI), and even games using Unity.
- **Object-Oriented Design:** C# encourages modular, maintainable code that scales well in large projects.
- **Rich Library Ecosystem:** The .NET framework provides libraries for virtually every use case, from database access to cryptography.
- **Asynchronous Programming:** Modern C# emphasizes async/await patterns for non-blocking, scalable applications.
- **Enterprise Integration:** Its compatibility with tools like Azure, SQL Server, and Microsoft Office makes C# a natural fit for enterprise solutions.

## C# vs. Other Languages

While JavaScript continues to dominate web development and Python excels in data science, C# truly stands out as an exceptionally powerful and versatile workhorse for enterprise, desktop, and game development. It strikes a near-perfect balance between raw performance and user accessibility, offering a more structured, type-safe alternative to Python, while being less verbose and much more efficient than Java.

For developers looking for a flexible, general-purpose language that can tackle a wide range of diverse and complex challenges with ease, precision, and scalability, C# isn't just a good option—it's the go-to choice. Whether you are building cutting-edge software or solving intricate problems, C# has proven itself to be a reliable and consistently high-performing language that outshines the competition. It's why it's my compiled language of choice.



## Limitations of C#

- **Learning Curve:** Its depth and flexibility can be overwhelming for beginners. This can be scary for some people.
- **Resource-Intensive:** Managed code can be slower than low-level languages like C or C++.
- **Platform-Specific Features:** Some APIs and libraries are limited to Windows environments.
- **Dependency on .NET:** C# applications rely heavily on the .NET runtime, which may not be preinstalled in some systems. Yes, you will get to experience this firsthand if you start creating and distributing compiled apps.

Regardless of the caveats, C#'s powerful combination of performance, scalability, and developer-friendly tools makes it a go-to choice for professionals building the future of software.

---

Fellow Scripters,

Don't shy away from C#. While it is a full fledged programming language with greater complexity and a steeper learning curve, mastering a compiled language brings significant advantages. By learning C#, you'll unlock access to a vast array of robust libraries and frameworks, empowering you to build custom tools you need—tools that may not exist anywhere else.

---



## C#: Design

---

<https://learn.microsoft.com/en-us/dotnet/csharp/>

**Getting things done with C#.** I have added a few complete code examples on what you can accomplish with C#—as far as desktop management goes. These are just the basics. C# is a powerful language, and can do much more than what is presented here. Perhaps, if I release a second edition (or an advanced book), we'll delve deeper into C#, specifically building utilities.

## Basic Construction

---

```
using System; // Allows access to types and utilities, like Console.WriteLine.  
using System.IO; // Enables file and directory operations.  
  
// A class is a blueprint for creating objects or grouping related methods and  
// data together.  
class GreetingScript  
{  
    // Main is the program's entry point, where execution begins.  
    public static void Main(string[] args)  
    {  
        // Set console title.  
        Console.Title = "Greeting Script";  
  
        // Set text and background colors (black background, green text)  
        Console.BackgroundColor = ConsoleColor.Black;  
        Console.ForegroundColor = ConsoleColor.Green;  
  
        // Clear the console window.  
        Console.Clear();  
  
        // Display welcome message.  
        Console.WriteLine("Welcome to C# Programming!\n");  
  
        // Prompt the user for their name.  
        Console.Write("What is your name? ");
```



```
string name = Console.ReadLine();

// Display a personalized greeting.
Console.WriteLine($"\\nHello, {name}! Have a great day!");

// Pause the script to allow the user to see the message
Console.WriteLine("\\nPress any key to continue...\\n");
Console.ReadKey();

}

} // Pay attention to brackets. Too few or too many can break your program.
```



## C#: Return User UPN

```
using System;
using Microsoft.Win32;

class Program
{
    static void Main()
    {
        // Define the registry keys and values to query.
        string rootKey1 =
"HKEY_LOCAL_MACHINE\\Software\\Microsoft\\Windows\\CurrentVersion\\Authentication\\LogonUI";
        string rootValue1 = "LoggedOnUserSID";

        string loggedOnUserSID = GetRegistryValue(rootKey1, rootValue1);

        if (!string.IsNullOrEmpty(loggedOnUserSID))
        {
            string rootKey2 =
$"HKEY_USERS\\{loggedOnUserSID}\\Software\\Microsoft\\Office\\16.0\\Common\\ServicesManagerCache\\Identities";
            string rootValue2 = "ConnectionUserUpn";

            string connectionUserUpn = GetRegistryValue(rootKey2, rootValue2);

            Console.Clear();
            Console.WriteLine(connectionUserUpn ?? "ConnectionUserUpn not found.");
        }
        else
        {
            Console.WriteLine("LoggedOnUserSID not found.");
        }

        Console.WriteLine("Press any key to exit...");
        Console.ReadKey();
    }

    static string GetRegistryValue(string rootKey, string rootValue)
    {
        try
        {
            // Open the base registry key.
```



```
using (RegistryKey baseKey =
RegistryKey.OpenBaseKey(GetRegistryHive(rootKey),
RegistryView.Default))
{
    // Parse the subkey path from the rootKey.
    string subKeyPath = rootKey.Substring(rootKey.IndexOf("\\\\") +
1);
    using (RegistryKey subKey = baseKey.OpenSubKey(subKeyPath))
    {
        if (subKey != null)
        {
            foreach (string subKeyName in subKey.GetSubKeyNames())
            {
                using (RegistryKey childKey =
subKey.OpenSubKey(subKeyName))
                {
                    if (childKey != null &&
childKey.GetValue(rootValue) is string value)
                    {
                        return value;
                    }

                    // Recursive call to search deeper.
                    string result =
GetRegistryValue($"{rootKey}\\{subKeyName}", rootValue);
                    if (!string.IsNullOrEmpty(result))
                    {
                        return result;
                    }
                }
            }
        }
    }
}
catch (Exception ex)
{
    Console.WriteLine($"Error scanning key: {rootKey} - {ex.Message}");
}

return null;
}

static RegistryHive GetRegistryHive(string rootKey)
{
```



```
if (rootKey.StartsWith("HKEY_LOCAL_MACHINE",
    StringComparison.OrdinalIgnoreCase))
    return RegistryHive.LocalMachine;

if (rootKey.StartsWith("HKEY_USERS",
    StringComparison.OrdinalIgnoreCase))
    return RegistryHive.Users;

throw new ArgumentException($"Unsupported root key: {rootKey}");
}
}
```



## C#: Return Username

```
using System;
using Microsoft.Win32;
using System.Diagnostics;

class Program
{
    static void Main()
    {
        string username = null;
        string basePath =
@"SOFTWARE\Microsoft\Windows\CurrentVersion\Authentication\LogonUI\SessionData";

        for (int session = 1; session <= 3; session++)
        {
            try
            {
                using (RegistryKey baseKey =
                    Registry.LocalMachine.OpenSubKey($"{{basePath}}\\{{session}}"))
                {
                    if (baseKey != null)
                    {
                        username = baseKey.GetValue("LoggedOnSAMUser") as
                        string;
                        if (!string.IsNullOrEmpty(username))
                        {
                            break;
                        }
                    }
                }
            }
            catch
            {
                // Ignore errors for non-existent paths or properties.
            }
        }

        string currentUser = null;

        if (string.IsNullOrEmpty(username))
        {
            // Attempt to use QUSER as a fallback.
            currentUser = GetActiveUserFromQUSER();
        }
    }
}
```



```
}

// If QUSER fails, parse from username.
if (string.IsNullOrEmpty(currentUser) &&
!string.IsNullOrEmpty(username))
{
    currentUser = username.Split('\\')[^1];
}

Console.Clear();
Console.WriteLine($"Domain User: {username}");
Console.WriteLine($"Parsed: {currentUser}");
Console.WriteLine();

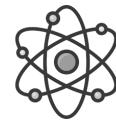
Console.WriteLine("Press any key to exit...");
Console.ReadKey();
}

static string GetActiveUserFromQUSER()
{
    try
    {
        Process quserProcess = new Process
        {
            StartInfo = new ProcessStartInfo
            {
                FileName = "cmd.exe",
                Arguments = "/c quser",
                RedirectStandardOutput = true,
                UseShellExecute = false,
                CreateNoWindow = true
            }
        };

        quserProcess.Start();
        string output = quserProcess.StandardOutput.ReadToEnd();
        quserProcess.WaitForExit();

        foreach (var line in output.Split(new[] { '\n', '\r' },
StringSplitOptions.RemoveEmptyEntries))
        {
            if (System.Text.RegularExpressions.Regex.IsMatch(line,
"\s+(\S+)\s+(\S+)\s+(\d+)\s+Active\s+"))
            {
                var match =
System.Text.RegularExpressions.Regex.Match(line,
"\s+(\S+)\s+(\S+)\s+(\d+)\s+Active\s+");

```



```
        return match.Groups[1].Value;
    }
}
catch
{
    // Ignore any errors from QUSER execution.
    // When the catch is empty like this, we call
    // it swallowing, or error suppression.
}

return null;
}
}
```



## C#: Return User SID

---

```
using System;
using Microsoft.Win32;

class Program
{
    static void Main()
    {
        // Define the registry key and value to query.
        string rootKey =
"HKEY_LOCAL_MACHINE\\SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Authentication\\LogonUI";
        string rootValue = "LoggedOnUserSID";

        string loggedOnUserSID = GetRegistryValue(rootKey, rootValue);

        Console.Clear();
        Console.WriteLine(loggedOnUserSID ?? "LoggedOnUserSID not found.");

        Console.WriteLine("Press any key to exit...");
        Console.ReadKey();
    }

    static string GetRegistryValue(string rootKey, string rootValue)
    {
        try
        {
            // Open the base registry key.
            using (RegistryKey baseKey =
RegistryKey.OpenBaseKey(GetRegistryHive(rootKey),
RegistryView.Default))
            {
                // Parse the subkey path from the rootKey.
                string subKeyPath = rootKey.Substring(rootKey.IndexOf("\\") +
1);
                using (RegistryKey subKey = baseKey.OpenSubKey(subKeyPath))
                {
                    if (subKey != null)
                    {
                        foreach (string subKeyName in subKey.GetSubKeyNames())
                        {
                            using (RegistryKey childKey =
subKey.OpenSubKey(subKeyName))
                            {

```



```

        if (childKey != null &&
            childKey.GetValue(rootValue) is string value)
        {
            return value;
        }

        // Recursive call to search deeper.
        string result =
            GetRegistryValue($"{rootKey}\\{subKeyName}",

rootValue);
        if (!string.IsNullOrEmpty(result))
        {
            return result;
        }
    }
}
}
}
}

catch (Exception ex)
{
    Console.WriteLine($"Error scanning key: {rootKey} - {ex.Message}");
}

return null;
}

static RegistryHive GetRegistryHive(string rootKey)
{
    if (rootKey.StartsWith("HKEY_LOCAL_MACHINE",
        StringComparison.OrdinalIgnoreCase))
        return RegistryHive.LocalMachine;

    if (rootKey.StartsWith("HKEY_USERS",
        StringComparison.OrdinalIgnoreCase))
        return RegistryHive.Users;

    throw new ArgumentException($"Unsupported root key: {rootKey}");
}

```



## C#: Launch & Kill Process

---

```
using System;
using System.Diagnostics;

class LaunchAndKillProcess
{
    // Define a timer function to handle the sleep.
    public static void Timer(int seconds)
    {
        Console.WriteLine("");
        // Convert seconds to milliseconds.
        System.Threading.Thread.Sleep(seconds * 1000);
    }

    public static void Main(string[] args)
    {
        Console.WriteLine("Launching Notepad..."); Timer(3);

        // Start the Notepad process.
        Process process = Process.Start("Notepad");
        // Or _ = Process.Start("Notepad");

        Console.WriteLine("Notepad launched!"); Timer(3);

        Console.WriteLine("Killing Notepad..."); Timer(3);

        // Kill any Notepad processes using foreach loop.
        foreach (Process runningProcess in
        Process.GetProcessesByName("Notepad"))
        {
            runningProcess.Kill();
        }

        Console.WriteLine("Notepad process terminated!"); Timer(3);

        // Wait for user input before exiting.
        Console.WriteLine("\nPress any key to exit...\n");
        Console.ReadKey();
    }
}
```



## C#: Read from Text File

---

```
using System;
using System.IO;

class ReadFileContents
{
    public static void Main(string[] args)
    {
        // Read the contents of our file.
        string contents = File.ReadAllText(@"C:\CSharp\test.txt");

        // Output contents to console.
        Console.WriteLine("Contents of file:\n" + contents);

        Console.WriteLine("\nPress any key to continue...\n");
        Console.ReadLine();
    }
}
```

## C#: Read from Text File, Line by Line

---

```
using System;
using System.IO;

class CreateAndReadFileContents
{
    public static void Main(string[] args)
    {
        string filePath = @"C:\CSharp\test.txt";

        // Create the directory if it does not exist.
        string directory = Path.GetDirectoryName(filePath);
        if (!Directory.Exists(directory))
        {
            Directory.CreateDirectory(directory);
        }

        // LAB SETUP
        // Contents to write to the file.
        string[] fileContents = new string[]
        {
            "Eddie Jackson",
            "Computer-Lab1",
        }
    }
}
```

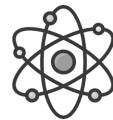


```
    "TYNEKJG-12"
};

// Write the contents to the file.
try
{
    File.WriteAllText(filePath, fileContents);
    Console.WriteLine("File created successfully with the following
content:");
    foreach (string line in fileContents)
    {
        Console.WriteLine(line);
    }
    Console.WriteLine("");
}
catch (Exception ex)
{
    Console.WriteLine($"An error occurred while creating the file:
{ex.Message}");
    return;
}

// Now read the file line by line and display it.
if (File.Exists(filePath))
{
    Console.WriteLine("Let's read the file line by line...");
    int lineNumber = 1;
    foreach (string line in File.ReadLines(filePath))
    {
        Console.WriteLine($"Line {lineNumber}: {line}");
        // add your code here
        lineNumber++;
    }
}
else
{
    Console.WriteLine("File not found.");
}

Console.WriteLine("\nPress any key to continue...\n");
Console.ReadLine();
}
}
```



## C#: Write to Text File

```
-----
using System;
using System.IO;

class WriteFileContents
{
    public static void Main(string[] args)
    {
        // Write to file using StreamWriter.
        string contentToWrite = "\nThis is another line of text to be written
        to the file.';

        using (StreamWriter writer = new StreamWriter(@"C:\CSharp\test.txt",
        append: true))
        {
            writer.WriteLine(contentToWrite);
        }

        // Read the contents of the file.
        string contents = File.ReadAllText(@"C:\CSharp\test.txt");

        // Output contents to console.
        Console.WriteLine("Contents of file:\n" + contents);

        Console.WriteLine("\nPress any key to continue...\n");
        Console.ReadLine();
    }
}
```



## 13. Logic

---

Logical operations are fundamental to programming, providing the backbone for decision-making and control flow within applications. These operations—**AND**, **OR**, **NOT**, and **XOR**—serve as powerful tools that allow developers to define conditions, evaluate expressions, and implement branching logic with precision. By manipulating Boolean values (**TRUE** or **FALSE**), logical operators enable programs to make informed decisions, respond to varying inputs, and execute specific tasks based on defined criteria.

The **AND** operator evaluates whether all conditions are true, ensuring that multiple requirements are satisfied before executing a block of code. The **OR** operator, on the other hand, checks whether at least one condition is true, providing flexibility in scenarios where only partial criteria need to be met. The **NOT** operator inverts Boolean values, allowing developers to reverse conditions and simplify logic in complex workflows. Finally, the **XOR** operator, or exclusive OR, ensures a condition is true only if the operands differ, making it valuable for toggling states or validating mutually exclusive events.

Logical operators also play a critical role in control structures like **IF STATEMENTS**, **LOOPS**, and **CONDITIONAL EXPRESSIONS**. Mastering their use ensures that developers can handle diverse programming challenges with clarity and precision. In this section, we'll explore the most common logical operators through simple, practical examples that highlight their functionality and demonstrate how they can be applied in real-world scenarios.

Let's review more about the logical operators.



## Logical Operators

---

### Our Operator List

#### AND ( && Operator )

---

```
bool a = true;  
bool b = false;  
bool result = a && b; // result is false
```

#### OR ( || Operator )

---

```
bool a = true;  
bool b = false;  
bool result = a || b; // result is true
```

#### NOT ( ! Operator )

---

```
bool a = true;  
bool result = !a; // result is false
```

#### XOR ( ^ Operator )

---

```
bool a = true;  
bool b = false;  
bool result = a ^ b; // result is true
```

If logical operators feel unrelatable, or you're struggling to make sense of how all this logic fits together, don't worry—help is on the way. In the next two sections we'll break down some common logical scenarios you might encounter and walk through clear, practical code examples to make these concepts easier to grasp.

Let's take a look.



## Common Logic Scenarios

---

Logical operators are fundamental tools used in programming to perform logical comparisons and control flow decisions.

Here's a breakdown of some common scenarios where each logical operator might be used.

### AND ( && Operator )

---

The AND operator is used when you want to check if multiple conditions are true at the same time. The result will only be true if all conditions evaluate to true.

Validating user input: If you want to check that a user has entered both a valid username and password:

#### C# Code

```
bool isUsernameValid = true;
bool isPasswordValid = true;

if (isUsernameValid && isPasswordValid) {
    // Proceed With Login
} else {
    // Show Error Message
}
```

Complex condition checking: When performing operations that require multiple conditions, such as checking if a user is both an administrator and has accepted the terms and conditions:

#### C# Code

```
bool isAdmin = true;
bool hasAcceptedTerms = false;
if (isAdmin && hasAcceptedTerms) {
```



```
// Allow Access to Admin Panel
} else {
    // Deny Access
}
```

## OR ( || Operator )

---

The OR operator is used when you want to check if at least one condition is true. The result is true if any of the conditions are true.

Checking multiple options: If you're checking if a user has permission based on different roles:

### C# Code

```
bool isAdmin = false;
bool isEditor = true;
if (isAdmin || isEditor) {
    // Allow Access Edit
} else {
    // Deny Access
}
```

Feature availability based on multiple conditions: If a feature is enabled for multiple reasons, such as being part of a certain group or having a specific privilege:

### C# Code

```
bool isPremiumMember = false;
bool hasCoupon = true;
if (isPremiumMember || hasCoupon) {
    // Provide Discount
} else {
    // Regular Price
}
```



## NOT ( ! Operator )

---

The NOT operator is used to invert a condition. If the condition is true, it becomes false, and if it is false, it becomes true.

Negating a condition: If you want to perform an action only if a condition is false, use the NOT operator:

### C# Code

```
bool isLoggedIn = false;  
if (!isLoggedIn) {  
    // Redirect To Login Page  
} else {  
    // Proceed With Dashboard  
}
```

Checking for the absence of a condition: If you're checking whether a user has not completed a form or a task:

### C# Code

```
bool isTaskComplete = false;  
if (!isTaskComplete) {  
    // Show Task Incomplete Message  
}
```

## XOR ( ^ Operator )

---

The XOR (exclusive OR) operator is used when you want to check if exactly one condition is true. The result is true if only one condition is true, but false if both are true or both are false.

Binary toggling: XOR is often used in scenarios where flipping between two states is required:



### C# Code

```
bool isActive = true;  
bool toggle = false;  
  
isActive = isActive ^ toggle; // isActive will become false -true XOR false.
```

Error detection or signaling: XOR can be used for certain error-checking algorithms where two values are compared to determine discrepancies:

### C# Code

```
bool input1 = true;  
bool input2 = false;  
  
bool errorDetected = input1 ^ input2;
```

Toggling features: If toggling between two mutually exclusive options (enabling and disabling a feature):

### C# Code

```
bool isEnabled = false;  
bool toggle = true;  
  
isEnabled = isEnabled ^ toggle; // isEnabled becomes true.
```



## Example Code

---

The C# code below illustrates the fundamental use of logical operators. Copy and paste into Visual Studio and compile, and the app will walk-through which logic operators are being used. Do try to understand each one.

```
using System;
using System.Threading; // Thread.Sleep

class LogicDemo
{
    static void Main()
    {
        // Imagine we are checking if someone is eligible to enter a club.
        bool hasTicket = true; // The person has a ticket.
        bool isMember = false; // The person is not a member.

        // AND Operator: Both conditions must be true.
        Console.WriteLine("Using the AND operator (&&):");

        Console.WriteLine($"Can enter with both ticket and membership?
{hasTicket && isMember}"); // false
Thread.Sleep(3000); // Wait for 3 seconds

        // OR Operator: Only one condition needs to be true.
        Console.WriteLine("\nUsing the OR operator (||):");

        Console.WriteLine($"Can enter with either a ticket or membership?
{hasTicket || isMember}"); // true
Thread.Sleep(3000); // Wait for 3 seconds

        // NOT Operator: Reverses the condition.
        Console.WriteLine("\nUsing the NOT operator (!):");

        Console.WriteLine($"Does not have a ticket? {!hasTicket}"); // false
Thread.Sleep(3000); // Wait for 3 seconds

        // XOR Operator: True only if one condition is true, but not both.
        Console.WriteLine("\nUsing the XOR operator (^):");
    }
}
```



```
Console.WriteLine($"Can enter with either a ticket or membership, but  
not both? {hasTicket ^ isMember}"); // true  
Thread.Sleep(3000); // Wait for 3 seconds  
  
// Prompt the user to press any key to exit.  
Console.WriteLine("\nPress any key to exit.");  
Console.ReadKey();  
}  
}
```



## 14. Sorting Algorithms

---

Sorting algorithms are the backbone of many computational tasks, enabling the efficient organization of data in ways that improve search, indexing, and retrieval operations. These algorithms arrange elements in a predetermined order—ascending or descending—based on a comparison criterion. The efficiency of sorting algorithms can have a significant impact on the overall performance of applications, from databases to search engines, and can often be a determining factor in the responsiveness of systems. In situations where time and resources are critical, choosing the right sorting algorithm can make a notable difference in execution speed and resource consumption.

Sorting has always been a key concern in computer science, particularly as data volumes grew and became more complex. In the early days of computing, simple algorithms like Bubble Sort and Selection Sort were developed to tackle basic sorting problems. While these algorithms were straightforward and easy to implement, they became inefficient as data sizes increased, often performing poorly on large datasets. With the rise of more complex systems and increased computational power, sorting algorithms like Quick Sort and Merge Sort emerged, bringing substantial performance improvements. These algorithms leveraged more sophisticated techniques, such as divide-and-conquer strategies, which allowed them to scale more effectively as data sizes grew.

Today, sorting has expanded beyond just comparing elements. Algorithms like Radix Sort and Counting Sort, which avoid direct comparisons, are tailored for specific types of data and offer specialized advantages in terms of speed and memory usage. These algorithms can outperform comparison-based algorithms in certain contexts, especially when dealing with large sets of data with limited range or certain properties. As datasets continue to grow in size and complexity, the ongoing evolution of sorting algorithms reflects the ever-increasing demand for efficiency, adaptability, and speed in handling large amounts of data across diverse platforms and applications.

Let's review some common search algorithms.



## Quick Sort

---

Quick Sort is a highly efficient sorting algorithm that employs the divide-and-conquer strategy. It works by selecting a "pivot" element from the array and partitioning the other elements into two groups: those less than the pivot and those greater than the pivot. The process is recursively applied to the sub-arrays formed by the partition. The key advantages of Quick Sort include its efficiency for large datasets and its average-case time complexity of  $O(n \log n)$ . However, in the worst case, its time complexity can degrade to  $O(n^2)$ , typically mitigated by choosing a good pivot (using randomization).

### How Quick Sort Works

#### Choose a Pivot

---

Select a pivot element from the array, which will be used to partition the array.

#### Partition the Array

---

Rearrange the elements so that elements smaller than the pivot are on the left and elements greater than the pivot are on the right.

#### Recursively Sort Subarrays

---

Recursively apply the same process to the subarrays formed by splitting the array into two parts: one with elements smaller than the pivot and one with elements larger.

#### End Condition

---

The algorithm completes when the subarrays contain only one element or are empty, meaning no further partitioning is possible.



## Time Complexity

### Best Case

$O(n \log n)$  — When the pivot divides the array into two equal halves at each step.

### Average Case

$O(n \log n)$  — Assuming random distribution of elements.

### Worst Case

$O(n^2)$  — When the pivot chosen is always the smallest or largest element, leading to highly unbalanced partitions.

## Space Complexity

$O(\log n)$  — Due to the recursive function calls.

## Real-World Use

- **Online Shopping:** Sorting product prices on an e-commerce website from lowest to highest for quick customer browsing.
- **Game Leaderboards:** Ranking player scores in multiplayer games rapidly during live updates.



## Example Code

---

### PowerShell

---

```
function Quick-Sort {
    param (
        [ref]$Elements,
        [int]$Left,
        [int]$Right
    )

    $i = $Left
    $j = $Right
    $Pivot = $Elements.Value[($Left + $Right) / 2]

    while ($i -le $j) {
        while ($Elements.Value[$i] -lt $Pivot) { $i++ }
        while ($Elements.Value[$j] -gt $Pivot) { $j-- }

        if ($i -le $j) {
            # Swap
            $Temp = $Elements.Value[$i]
            $Elements.Value[$i] = $Elements.Value[$j]
            $Elements.Value[$j] = $Temp

            $i++
            $j--
        }
    }

    # Recursive Calls
    if ($Left -lt $j) {
        Quick-Sort -Elements $Elements -Left $Left -Right $j
    }

    if ($i -lt $Right) {
        Quick-Sort -Elements $Elements -Left $i -Right $Right
    }
}

# Example Usage
$arrElements = @("z", "e", "x", "c", "m", "q", "a")
```



```
Clear-Host
```

```
# Display the unsorted array.  
Write-Host "Unsorted array:"  
Write-Host ($arrElements -Join ", ")  
Write-Host ("`n")  
  
# Sort the array  
Quick-Sort -Elements ([ref]$arrElements) -Left 0 -Right ($arrElements.Length - 1)  
  
# Display the sorted array.  
Write-Host "Sorted array:"  
Write-Host ($arrElements -Join ", ") # Display the sorted array.  
Write-Host ("`n"); Read-Host
```

### | output |

```
Unsorted array:  
z, e, x, c, m, q, a
```

```
Sorted array:  
a, c, e, m, q, x, z
```



## C#

---

```
using System;

namespace Quicksort
{
    class Program
    {
        static void Main(string[] args)
        {
            // Create an arrElements array of string elements.
            string[] arrElements = { "z", "e", "x", "c", "m", "q", "a" };

            // Display the arrElements array.
            Console.WriteLine("Unsorted array:");
            for (int i = 0; i < arrElements.Length; i++)
            {
                Console.Write(arrElements[i] + " ");
            }

            Console.WriteLine("\n");

            // Sort the array.
            Quicksort(arrElements, 0, arrElements.Length - 1);

            // Display the sorted array.
            Console.WriteLine("Sorted array:");
            for (int i = 0; i < arrElements.Length; i++)
            {
                Console.Write(arrElements[i] + " ");
            }

            Console.WriteLine();

            Console.ReadLine(); // Wait for a keypress.
        }

        public static void Quicksort(IComparable[] elements, int left, int right)
        {
            int i = left, j = right;
            IComparable pivot = elements[(left + right) / 2];

            while (i <= j)
```



```
{  
    while (elements[i].CompareTo(pivot) < 0) { i++; } while  
    (elements[j].CompareTo(pivot) > 0)  
    {  
        j--;  
    }  
  
    if (i <= j)  
    {  
        // Swap  
        IComparable tmp = elements[i];  
        elements[i] = elements[j];  
        elements[j] = tmp;  
  
        i++;  
        j--;  
    }  
}  
  
// Recursive Calls  
if (left < j)  
{  
    Quicksort(elements, left, j);  
}  
  
if (i < right)  
{  
    Quicksort(elements, i, right);  
}  
}  
}  
}  
}
```

### | output |

Unsorted array:  
z e x c m q a

Sorted array:  
a c e m q x z



## Merge Sort

---

Merge Sort is a stable and efficient sorting algorithm that uses the divide-and-conquer strategy. It divides the input array into two halves, recursively sorts each half, and then merges the sorted halves to produce the final sorted array. This process ensures that the algorithm always has a time complexity of  $O(n \log n)$  for all cases (best, average, and worst). Merge Sort requires additional memory for the temporary arrays used during merging, making it less space-efficient than in-place algorithms.

### How Merge Sort Works

#### Divide the Array

---

Split the array into two halves until each subarray contains only one element.

#### Recursively Sort Each Half

---

Recursively apply the Merge Sort algorithm to the left and right halves of the array.

#### Merge the Sorted Halves

---

Merge the two sorted subarrays by comparing their elements and placing them in a new array in sorted order.

#### End Condition

---

The algorithm completes when all subarrays are merged back together into a single sorted array.



## Time Complexity

### Best Case

$O(n \log n)$  — When the input array is already sorted or nearly sorted.

### Average Case

$O(n \log n)$  — Since the process of dividing the array and merging back together is consistent across all inputs.

### Worst Case

$O(n \log n)$  — Which is the same as the average and best case.

## Space Complexity

$O(n)$  — Because Merge Sort requires additional space for the temporary arrays used for merge.

## Real-World Use

- **Merging Playlists:** Combining two sorted playlists into one unified playlist while keeping the songs in order of release date.
- **Sorting Contacts:** Organizing names in a contact list alphabetically after merging contacts from multiple devices.



## Example Code

---

### PowerShell

---

```
function Merge-Sort {
    param (
        [int[]]$Array
    )

    # Base case: if the array has 1 or 0 elements, it is already sorted.
    if ($Array.Length -le 1) {
        return $Array
    }

    # Find the midpoint of the array.
    $Mid = [math]::Floor($Array.Length / 2)

    # Divide the array into left and right halves.
    $leftArray = if ($Mid -gt 0) { $Array[0..($Mid - 1)] } else { @() }
    $rightArray = if ($Mid -lt $Array.Length) { $Array[$Mid..($Array.Length - 1)] } else { @() }

    # Recursively sort both halves.
    $sortedLeft = Merge-Sort -Array $leftArray
    $sortedRight = Merge-Sort -Array $rightArray

    # Merge the sorted halves.
    return Merge-Arrays -LeftArray $sortedLeft -RightArray $sortedRight
}

function Merge-Arrays {
    param (
        [int[]]$leftArray,
        [int[]]$rightArray
    )

    $Result = @()

    # Compare elements from both arrays and build the result array.
    while ($leftArray.Count -gt 0 -and $rightArray.Count -gt 0) {
        if ($leftArray[0] -le $rightArray[0]) {
            # Add the smaller element from the left array.
            $Result += $leftArray[0]
```



```
$leftArray = if ($leftArray.Count -gt 1) {
    $leftArray[1..($leftArray.Count - 1)] } else { @() }
} else {
    # Add the smaller element from the right array.
    $Result += $rightArray[0]
    $rightArray = if ($rightArray.Count -gt 1) {
        $rightArray[1..($rightArray.Count - 1)] } else { @() }
}
}

# Add any remaining elements from the left array.
if ($leftArray.Count -gt 0) {
    $Result += $leftArray
}

# Add any remaining elements from the right array.
if ($rightArray.Count -gt 0) {
    $Result += $rightArray
}

return $Result
}

Clear-Host

# Example Usage
$arrElements = @(2, 5, -4, 11, 0, 18, 22, 67, 51, 6, 19, 13)

# Display Unsorted Array
Write-Host "Unsorted Array:`n $($arrElements -Join ', ')"
Write-Host "`n"

$Sorted = Merge-Sort -Array $arrElements
# Display Sorted Array
Write-Host "Sorted Array:`n $($Sorted -Join ', ')" # Display the sorted array.
Write-Host "`n"; Read-Host
```

## | output |

Unsorted Array:  
2, 5, -4, 11, 0, 18, 22, 67, 51, 6, 19, 13

Sorted Array:  
-4, 0, 2, 5, 6, 11, 13, 18, 19, 22, 51, 67



## C#

---

```
using System;

class MergeSort
{
    // The Sort method is responsible for sorting an array using the Merge
    // Sort algorithm. It recursively divides the array into two halves and
    // merges them in sorted order.
    public static void Sort(int[] array)
    {
        // Base case: if array has 1 or fewer elements, it is already sorted.
        if (array.Length <= 1)
            return;

        // Find the middle index of the array.
        int mid = array.Length / 2;

        // Create temporary arrays for the left and right halves of the array.
        int[] left = new int[mid];
        int[] right = new int[array.Length - mid];

        // Copy the left half of the array into the left temporary array.
        Array.Copy(array, 0, left, 0, mid);

        // Copy the right half of the array into the right temporary array.
        Array.Copy(array, mid, right, 0, array.Length - mid);

        // Recursively sort the left and right sub-arrays.
        Sort(left);
        Sort(right);

        // Merge sorted left and right sub-arrays back into original array.
        Merge(array, left, right);
    }

    // The Merge method merges two sorted arrays (left and right) into
    // a single sorted array.
    private static void Merge(int[] array, int[] left, int[] right)
    {
        int i = 0, j = 0, k = 0;

        // Merge elements from both left and right arrays while both
        // have elements.
```



```
while (i < left.Length && j < right.Length)
{
    // If the current element in the left array is smaller or equal,
    // add it to the result array.
    if (left[i] <= right[j])
    {
        array[k++] = left[i++];
    }
    else
    {
        // Otherwise, add the current element from the right array.
        array[k++] = right[j++];
    }
}

// If there are remaining elements in the left array, add them to
// the result.
while (i < left.Length)
{
    array[k++] = left[i++];
}

// If there are remaining elements in the right array, add them to
// the result.
while (j < right.Length)
{
    array[k++] = right[j++];
}

// The Main method is the entry point of the program and demonstrates how
// to use the MergeSort algorithm.
public static void Main(string[] args)
{
    // Example array to be sorted.
    int[] arrElements = { 34, 7, 23, 32, 5, 62 };

    // Display the original unsorted array.
    Console.WriteLine("Unsorted Array:\n" + string.Join(", ", arrElements));
    Console.WriteLine("");

    // Call the Sort method to sort the array.
    Sort(arrElements);

    // Display the sorted array.
    Console.WriteLine("Sorted Array:\n" + string.Join(", ", arrElements));
}
```



```
        Console.ReadLine(); // Wait for a keypress.  
    }  
}
```

### | output |

```
Unsorted Array:  
34, 7, 23, 32, 5, 62  
  
Sorted Array:  
5, 7, 23, 32, 34, 62
```



## Heap Sort

---

Heap Sort is a comparison-based sorting algorithm that uses a binary heap data structure. It first transforms the input array into a max-heap (where the largest element is at the root) and then repeatedly extracts the largest element, placing it at the end of the array while adjusting the heap. This process continues until the entire array is sorted.

Heap Sort has a time complexity of  $O(n \log n)$  in all cases, as each "heapify" operation (adjusting the heap) runs in  $O(\log n)$  and is applied to all elements. It is not a stable sort, but it is in-place, requiring only a constant amount of additional space.

## How Heap Sort Works

### Build a Max Heap

---

Arrange the elements in a binary tree structure to satisfy the max heap property, where each parent node is greater than its children.

### Extract the Root (Maximum Element)

---

Remove the root element (maximum value) of the heap and swap it with the last element in the heap.

### Re-heapify

---

Restore the heap property by "heapifying" the remaining elements. This involves comparing the new root with its children and swapping if necessary, then recursively heapifying the affected subtree.

### Repeat Extraction

---

Repeat the process of extracting the root and re-heapifying until all elements are sorted.

### End Condition

---

The algorithm completes when all elements are extracted from the heap and placed in their correct positions in the array.



## Time Complexity

### Best Case

$O(n \log n)$  — When the input array is already a valid heap.

### Average Case

$O(n \log n)$  — Because heap operations (insertion and extraction) take  $O(\log n)$  time, and we perform these operations  $n$  times.

### Worst Case

$O(n \log n)$  — Because the worst-case scenario still involves  $n$  extractions and reheapifications, and each heap operation (insertion or extraction) takes  $O(\log n)$  time.

## Space Complexity

$O(1)$  — Because Heap Sort is an in-place sorting algorithm.

## Real-World Use

- **Scheduling Tasks:** Organizing tasks by priority levels to ensure the most urgent tasks are completed first.
- **Processing Print Jobs:** Sorting print queue requests by file size or urgency before sending them to the printer.



## Example Code

---

### PowerShell

---

```
function Print-Array {
    param (
        [Parameter(Mandatory)]
        [array]$Array
    )
    # displays the elements of the array as a space-separated string.
    Write-Host ($Array -Join ' ')
}

function Swap-Elements {
    param (
        [Parameter(Mandatory)]
        [array]$Array,
        [int]$Pos1,
        [int]$Pos2
    )
    # Swaps two elements in the array.
    $temp = $Array[$Pos1]
    $Array[$Pos1] = $Array[$Pos2]
    $Array[$Pos2] = $temp
}

function Get-LeftChildPos {
    param (
        [int]$parentPos
    )
    # Returns the index of the left child of the given parent node.
    return (2 * ($parentPos + 1) - 1)
}

function Get-RightChildPos {
    param (
        [int]$parentPos
    )
    # Returns the index of the right child of the given parent node.
    return (2 * ($parentPos + 1))
}
```



```
function Sink-Element {
    param (
        [Parameter(Mandatory)]
        [array]$Array,
        [int]$heapSize,
        [int]$toSinkPos
    )
    # Ensures the heap property by sinking the element at the given position.
    $leftChildPos = Get-LeftChildPos -ParentPos $toSinkPos
    if ($leftChildPos -ge $heapSize) {
        return # If no left child exists, the node is a leaf.
    }

    $rightChildPos = Get-RightChildPos -ParentPos $toSinkPos
    $largestChildPos = $leftChildPos

    # Determine which child is larger (if 'right' child exists).
    if ($rightChildPos -lt $heapSize -and $Array[$rightChildPos] -gt
        $Array[$leftChildPos]) {
        $largestChildPos = $rightChildPos
    }

    # Swap if the largest child is greater than the current node.
    if ($Array[$largestChildPos] -gt $Array[$toSinkPos]) {
        Swap-Elements -Array $Array -Pos1 $toSinkPos -Pos2 $largestChildPos
        # Recursively sink the affected child.
        Sink-Element -Array $Array -HeapSize $heapSize -ToSinkPos
        $largestChildPos
    }
}

function Build-MaxHeap {
    param (
        [Parameter(Mandatory)]
        [array]$Array
    )
    # Builds a Max-Heap from the input array.
    $heapSize = $Array.Length
    for ($i = [math]::Floor($heapSize / 2) - 1; $i -ge 0; $i--) {
        Sink-Element -Array $Array -HeapSize $heapSize -ToSinkPos $i
    }
}

function Heap-Sort {
    param (
        [Parameter(Mandatory)]

```



```
[array]$Array
)
# Sorts the array using the Heap Sort algorithm.
Build-MaxHeap -Array $Array
$heapSize = $Array.Length

for ($i = $heapSize - 1; $i -ge 1; $i--) {
    # Swap the root with the last element.
    Swap-Elements -Array $Array -Pos1 0 -Pos2 $i
    $heapSize--
    # Restore the heap property.
    Sink-Element -Array $Array -HeapSize $heapSize -ToSinkPos 0
}
}

# Define the array to be sorted.
$array = @(2, 5, -4, 11, 0, 18, 22, 67, 51, 6) # Display the sorted array.

Clear-Host

# Display Unsorted Array
Write-Host "Unsorted Array:"
Print-Array -Array $array

Heap-Sort -Array $array # Perform the heap sort.

# Display Sorted Array
Write-Host "`nSorted Array:"
Print-Array -Array $array; Read-Host
```

## | output |

```
Unsorted Array:
2 5 -4 11 0 18 22 67 51 6

Sorted Array:
-4 0 2 5 6 11 18 22 51 67
```



## C#

---

```
using System;

namespace HeapSort
{
    public class MainClass
    {
        public static void Main(string[] args)
        {
            // Define an array of integers to be sorted.
            int[] arrElements = new int[] { 2, 5, -4, 11, 0, 18, 22, 67, 51, 6
            };

            // Display the original unsorted array.
            Console.WriteLine("Unsorted Array:");
            printArray(arrElements);

            // Sort the array using Heap Sort.
            heapSort(arrElements);

            // Display the sorted array
            Console.WriteLine("\n\nSorted Array:");
            printArray(arrElements);
            Console.WriteLine("\n");
            Console.ReadLine();
        }

        // Implements the Heap Sort algorithm.
        private static void heapSort<T>(T[] array) where T : IComparable<T>
        {
            int heapSize = array.Length;
            // Build the initial Max-Heap from the input array.
            buildMaxHeap(array);

            // Repeatedly extract the maximum element and rebuild the heap.
            for (int i = heapSize - 1; i >= 1; i--)
            {
                // Swap the root (maximum element) with the last element.
                swap(array, i, 0);

                // Reduce the heap size and sink the new root to maintain
                // heap properties.
                heapSize--;
                sink(array, heapSize, 0);
            }
        }

        // Swaps two elements in an array.
        private static void swap(T[] array, int i, int j)
        {
            T temp = array[i];
            array[i] = array[j];
            array[j] = temp;
        }

        // Sinks the element at index i to its correct position in the heap.
        private static void sink(T[] array, int heapSize, int i)
        {
            int largest = i;
            int left = 2 * i + 1;
            int right = 2 * i + 2;

            if (left < heapSize && array[left].CompareTo(array[largest]) > 0)
            {
                largest = left;
            }

            if (right < heapSize && array[right].CompareTo(array[largest]) > 0)
            {
                largest = right;
            }

            if (largest != i)
            {
                swap(array, i, largest);
                sink(array, heapSize, largest);
            }
        }

        // Prints the elements of an array.
        private static void printArray(T[] array)
        {
            foreach (T element in array)
            {
                Console.Write(element + " ");
            }
            Console.WriteLine();
        }
    }
}
```



```
        }

    }

    // Converts an array into a Max-Heap.
    private static void buildMaxHeap<T>(T[] array) where T :
        IComparable<T>
    {
        int heapSize = array.Length;

        // Start from the last non-leaf node and sink each element.
        for (int i = heapSize / 2 - 1; i >= 0; i--)
        {
            sink(array, heapSize, i);
        }
    }

    // Ensures the heap property by sinking the element at the
    // given position.
    private static void sink<T>(T[] array, int heapSize, int toSinkPos)
    where T : IComparable<T>
    {
        // If the current node has no left child, it's a leaf and
        // cannot sink further.
        if (getLeftChildPos(toSinkPos) >= heapSize)
        {
            return;
        }

        int largestChildPos;
        bool leftIsLargest;

        // Determine which child (left or right) is larger.
        if (getRightChildPos(toSinkPos) >= heapSize ||
array[getRightChildPos(toSinkPos)].CompareTo(array[getLeftChildPos(toSinkPos)] ) < 0)
        {
            largestChildPos = getLeftChildPos(toSinkPos);
            leftIsLargest = true;
        }
        else
        {
            largestChildPos = getRightChildPos(toSinkPos);
            leftIsLargest = false;
        }

        // Swap if the largest child is greater than the current node.
        if (array[largestChildPos].CompareTo(array[toSinkPos]) > 0)
```



```
{  
    swap(array, toSinkPos, largestChildPos);  
  
    // Recursively sink the affected child.  
    if (leftIsLargest)  
    {  
        sink(array, heapSize, getLeftChildPos(toSinkPos));  
    }  
    else  
    {  
        sink(array, heapSize, getRightChildPos(toSinkPos));  
    }  
}  
}  
  
// Swaps two elements in the array.  
private static void swap<T>(T[] array, int pos0, int pos1)  
{  
    T tmpVal = array[pos0];  
    array[pos0] = array[pos1];  
    array[pos1] = tmpVal;  
}  
  
// Returns the index of the left child of a given node.  
private static int getLeftChildPos(int parentPos)  
{  
    return 2 * (parentPos + 1) - 1;  
}  
  
// Returns the index of the right child of a given node.  
private static int getRightChildPos(int parentPos)  
{  
    return 2 * (parentPos + 1);  
}  
  
// displays the elements of the array.  
private static void printArray<T>(T[] array)  
{  
    foreach (T t in array)  
    {  
        Console.Write(t.ToString() + ' ');  
    }  
}  
}
```



## | output |

Unsorted Array:

2 5 -4 11 0 18 22 67 51 6

Sorted Array:

-4 0 2 5 6 11 18 22 51 67



## Insertion Sort

---

Insertion Sort is a simple comparison-based sorting algorithm that builds the final sorted array one element at a time. It works by iteratively taking an element from the unsorted portion of the array and inserting it into the correct position within the sorted portion.

### How Insertion Sort Works

#### Build a Max Heap

---

Arrange the elements in a binary tree structure to satisfy the max heap property, where each parent node is greater than its children.

#### Extract the Root (Maximum Element)

---

Remove the root element (maximum value) of the heap and swap it with the last element in the heap.

#### Re-heapify

---

Restore the heap property by "heapifying" the remaining elements. This involves comparing the new root with its children and swapping if necessary, then recursively heapifying the affected subtree.

#### Repeat Extraction

---

Repeat the process of extracting the root and re-heapifying until all elements are sorted.

#### End Condition

---

The algorithm completes when all elements are extracted from the heap and placed in their correct positions in the array.



## Time Complexity

### Best Case

$O(n)$  — When the array is already sorted.

### Average Case

$O(n^2)$  — When the array is unordered.

### Worst Case

$O(n^2)$  — When the array is in reverse order.

## Space Complexity

$O(1)$  — Because Insertion Sort is an in-place sorting algorithm.

## Real-World Use

- **Hand of Cards:** Sorting playing cards in a hand by suit or value during a game.
- **Classroom Assignments:** Arranging student names in alphabetical order as they are added to a list.



## Example Code

---

### PowerShell

---

```
# Function to perform insertion sort on an array.
function Insertion-Sort {
    param (
        [Parameter(Mandatory)]
        [array]$Array
    )
    # Loop through the array starting from the second element.
    for ($i = 1; $i -lt $Array.Length; $i++) {
        # Store the current element to be compared.
        $Key = $Array[$i]
        # Initialize the index of the previous element.
        $j = $i - 1

        # Shift elements of $Array[0..$i-1], that are greater than $key,
        # one position to the right.
        while ($j -ge 0 -and $Array[$j] -gt $Key) {
            $Array[$j + 1] = $Array[$j]
            $j--
        }
        # Insert the $key at its correct position.
        $Array[$j + 1] = $Key
    }
    return $Array # Return the sorted array.
}

# Function to display the elements of an array.
function Print-Array {
    param (
        [Parameter(Mandatory)]
        [array]$Array
    )
    # Join the array elements with spaces and display them.
    Write-Host ($Array -Join ' ')
}

Clear-Host
```



```
# Define and initialize an array of integers to be sorted.  
$arrElements = @(2, 5, -4, 11, 0, 18, 22, 67, 51, 6)  
  
Write-Host "Unsorted Array:"  
Print-Array -Array $arrElements # Display the original array.  
  
# Perform the insertion sort.  
$SortedArray = Insertion-Sort -Array $arrElements  
  
Write-Host "`nSorted Array:"  
Print-Array -Array $SortedArray # Display the sorted array.  
  
Write-Host "`n"; Read-Host
```

### | output |

```
Unsorted Array:  
2 5 -4 11 0 18 22 67 51 6  
  
Sorted Array:  
-4 0 2 5 6 11 18 22 51 67
```



## C#

---

```
using System;

namespace CommonInsertion_Sort
{
    class Program
    {
        static void Main(string[] args)
        {
            // Define and initialize an array of integers to be sorted.
            int[] arrElements = new int[10] { 2, 5, -4, 11, 0, 18, 22, 67, 51,
            6 };

            // Display the original array elements.
            Console.WriteLine("Unsorted Array:");
            PrintIntegerArray(arrElements);

            // Perform insertion sort and display the sorted array elements.
            Console.WriteLine("\n\nSorted Array:");
            PrintIntegerArray(InsertionSort(arrElements));

            // Wait for user input before closing
            Console.WriteLine("\n");
            Console.ReadLine();
        }

        // Sorts an array of integers using the insertion sort algorithm.
        static int[] InsertionSort(int[] inputArray)
        {
            // Iterate through the array starting from the second element.
            for (int i = 0; i < inputArray.Length - 1; i++)
            {
                // Compare the current element with its predecessors.
                for (int j = i + 1; j > 0; j--)
                {
                    // If the current element is smaller than its predecessor,
                    // swap them.
                    if (inputArray[j - 1] > inputArray[j])
                    {
                        int temp = inputArray[j - 1];
                        inputArray[j - 1] = inputArray[j];
                        inputArray[j] = temp;
                    }
                }
            }
        }
    }
}
```



```
        }

        return inputArray; // Return the sorted array.
    }

    // Displays the elements of an integer array.
    public static void PrintIntegerArray(int[] array)
    {
        // Loop through the array and display each element.
        foreach (int i in array)
        {
            Console.WriteLine(i.ToString() + " ");
        }
    }

    // Sorts an array of integers using insertion sort with a shifting
    // mechanism.
    public static int[] InsertionSortByShift(int[] inputArray)
    {
        // Iterate through the array starting from the second element.
        for (int i = 0; i < inputArray.Length - 1; i++)
        {
            int j;

            // Store the value to be inserted.
            var insertionValue = inputArray[i];

            // Shift elements to the right to make room for the
            // insertion value.
            for (j = i; j > 0; j--)
            {
                if (inputArray[j - 1] > insertionValue)
                {
                    inputArray[j] = inputArray[j - 1];
                }
            }
            // Insert the value at the correct position.
            inputArray[j] = insertionValue;
        }
        return inputArray; // Return the sorted array.
    }
}
```



## | output |

Unsorted Array:

2 5 -4 11 0 18 22 67 51 6

Sorted Array:

-4 0 2 5 6 11 18 22 51 67



## Bubble Sort

---

Bubble Sort is a simple comparison-based sorting algorithm that repeatedly steps through the list, compares adjacent elements, and swaps them if they are in the wrong order. This process is repeated until the list is sorted. Is this very efficient? Not really, but it can be used for smaller arrays.

### How Bubble Sort Works

#### Compare Adjacent Elements

---

Start from the first element and compare it with the next element in the array.

#### Swap if Out of Order

---

If the current element is greater than the next, swap them.

#### Move to Next Pair

---

Move to the next pair of adjacent elements and repeat the comparison and swapping if necessary.

#### Repeat for All Elements

---

Continue this process for all elements, reducing the range after each full pass, as the largest elements "bubble" to the end.

#### End Condition

---

The algorithm completes when a full pass is made without any swaps, indicating that the array is sorted.



## Time Complexity

### Best Case

$O(n)$  — When the array is already sorted (with an optimized version that stops early).

### Average Case

$O(n^2)$  — Because the algorithm compares each pair of elements in the array and performs swaps if needed.

### Worst Case

$O(n^2)$  — When the array is in reverse order.

## Space Complexity

$O(1)$  — Because Bubble Sort is an in-place sorting algorithm.

## Real-World Use

- **Sorting Scores:** Organizing test scores for a small class of students to identify the highest and lowest performers.
- **Arranging Keys:** Sorting a small number of house keys by size for easier identification.



## Example Code

---

### PowerShell

---

```
# Function to perform bubble sort on an array.
function Bubble-Sort {
    param (
        [Parameter(Mandatory)]
        [array]$Array
    )

    # Perform the bubble sort.
    # Outer loop for passes.
    for ($p = 0; $p -lt $Array.Length - 1; $p++) {
        # Inner loop for comparisons.
        for ($i = 0; $i -lt $Array.Length - 1; $i++) {
            # Compare Adjacent Elements
            if ($Array[$i] -gt $Array[$i + 1]) {
                # Swap elements if they are in the wrong order
                $temp = $Array[$i + 1]
                $Array[$i + 1] = $Array[$i]
                $Array[$i] = $temp
            }
        }
    }

    return $Array # Return the sorted array.
}

# Function to display the elements of an array.
function Print-Array {
    param (
        [Parameter(Mandatory)]
        [array]$Array
    )
    # Join the array elements with spaces and display them.
    Write-Host ($Array -Join ' ')
}

Clear-Host
```



```
# Define and initialize an array of integers to be sorted.  
$arrElements = @(3, 0, 2, 5, -1, 4, 1)  
  
Write-Host "Unsorted Array:"  
Print-Array -Array $arrElements # Display the original array.  
  
# Perform the bubble sort.  
$sortedArray = Bubble-Sort -Array $arrElements  
  
Write-Host "`nSorted Array:"  
Print-Array -Array $sortedArray # Display the sorted array.  
  
Write-Host "`n"; Read-Host
```

### | output |

Unsorted Array:  
3 0 2 5 -1 4 1

Sorted Array:  
-1 0 1 2 3 4 5



## C#

---

```
using System;

public class Bubble_Sort
{
    public static void Main(string[] args)
    {
        // Initialize an array of integers to be sorted.
        int[] a = { 3, 0, 2, 5, -1, 4, 1 };

        // Temporary Variable Swapping
        int t;

        // Display the unsorted array.
        Console.WriteLine("\nUnsorted array: ");
        foreach (int aa in a)
            Console.Write(aa + " ");

        // Perform bubble sort
        for (int p = 0; p <= a.Length - 2; p++) // Outer Loop Passes
        {
            // Inner loop for comparisons.
            for (int i = 0; i <= a.Length - 2; i++)
            {
                // Compare the adjacent elements.
                if (a[i] > a[i + 1])
                {
                    // Swap elements if they are in the wrong order.
                    t = a[i + 1];
                    a[i + 1] = a[i];
                    a[i] = t;
                }
            }
        }

        // Display the sorted array.
        Console.WriteLine("\n\nSorted array: ");
        foreach (int aa in a)
            Console.Write(aa + " ");
        Console.WriteLine("\n");

        // Wait for user input before exiting (optional for some consoles).
        Console.Read();
    }
}
```



## | output |

Unsorted Array:  
3 0 2 5 -1 4 1

Sorted Array:  
-1 0 1 2 3 4 5



## Counting Sort

---

Counting Sort is a non-comparison-based sorting algorithm that sorts integers by counting the occurrences of each unique element and using these counts to determine the positions of elements in the sorted array. It is particularly effective for sorting integers when the range of values (difference between the smallest and largest numbers) is not significantly larger than the size of the array.

### How Counting Sort Works

#### Find the Range

---

Identify the minimum and maximum values in the input array to determine the range of values.

#### Initialize Count Array

---

Create a count array with a size equal to the range of values, where each index represents the frequency of a particular element in the input array.

#### Count Frequencies

---

Iterate through the input array and increment the corresponding index in the count array based on each element's value.

#### Cumulative Count

---

Modify the count array by converting it to a cumulative count, where each index represents the position of the element in the sorted array.

#### Place Elements in Sorted Order

---

Using the cumulative count array, place the elements from the input array into their correct positions in the output array.



## End Condition

The algorithm completes when all elements are placed in the output array, resulting in a sorted array.

## Time Complexity

### Best Case

$O(n + k)$  — Where **n** is the number of elements and **k** is the range of input values (difference between the maximum and minimum values).

### Average Case

$O(n + k)$  — Where **n** is the number of elements and **k** is the range of input values.

### Worst Case

$O(n + k)$  — Where **n** is the number of elements and **k** is the range of input values.

## Space Complexity

$O(n + k)$  — Counting Sort requires extra space for the output array (**size n**) and the counting array (**size k**), where **n** is the number of elements to be sorted and **k** is the range of input values (the difference between the maximum and minimum values).

## Real-World Use

- **Counting Votes:** Tallying and sorting votes in an election by candidate ID, assuming the IDs fall within a specific range.
- **Inventory Counts:** Organizing stock items by quantity in a small range, like sorting items by number of units available.



## Example Code

---

### PowerShell

---

```
function Counting-Sort {
    # Define the input array.
    $array = @(3, 4, -2, 10, 0, 7, 21, 66, 50, 5, 1, 6)

    # Display the unsorted array.
    Write-Output "Unsorted array:"
    $array -Join ' '

    # Initialize the sorted array.
    $sortedArray = @($null) * $array.Length

    # Find the smallest and largest value in the array.
    $minVal = $array[0]
    $maxVal = $array[0]
    for ($i = 1; $i -lt $array.Length; $i++) {
        if ($array[$i] -lt $minVal) {
            $minVal = $array[$i]
        } elseif ($array[$i] -gt $maxVal) {
            $maxVal = $array[$i]
        }
    }

    # Initialize the frequency array.
    $counts = @()
    for ($i = 0; $i -le ($maxVal - $minVal); $i++) {
        $counts += 0
    }

    # Fill the frequency array.
    foreach ($num in $array) {
        $counts[$num - $minVal]++
    }

    # Adjust the frequency array to store cumulative counts.
    $counts[0]-- # Decrement the first element to make it 0-indexed.
    for ($i = 1; $i -lt $counts.Length; $i++) {
        $counts[$i] += $counts[$i - 1]
```



```
}

# Place the elements in sorted order in the output array.
for ($i = $array.Length - 1; $i -ge 0; $i--) {
    $sortedArray[$counts[$array[$i] - $minVal]--] = $array[$i]
}

# Display the sorted array.
Write-Output "`nSorted array:"
$sortedArray -Join ' ' ; Read-Host
}

# Run the function.
Counting-Sort
```

### | output |

```
Unsorted Array:
3 4 -2 10 0 7 21 66 50 5 1 6

Sorted Array:
-2 0 1 3 4 5 6 7 10 21 50 66
```



## C#

---

```
using System;

public class Counting_Sort
{
    public static void Main()
    {
        int[] array = new int[12] { 3, 4, -2, 10, 0, 7, 21, 66, 50, 5, 1, 6 };

        // Display the unsorted array.
        Console.WriteLine("Unsorted array: ");
        foreach (int num in array)
            Console.Write(num + " ");

        int[] sortedArray = new int[array.Length];

        // Find the smallest and largest value in the array.
        int minValue = array[0];
        int maxValue = array[0];
        for (int i = 1; i < array.Length; i++)
        {
            if (array[i] < minValue)
                minValue = array[i];
            else if (array[i] > maxValue)
                maxValue = array[i];
        }

        // Initialize the frequency array.
        int[] counts = new int[maxValue - minValue + 1];

        // Fill the frequency array
        for (int i = 0; i < array.Length; i++)
        {
            counts[array[i] - minValue]++;
        }

        // Adjust the frequency array to store cumulative counts.
        counts[0]--; // Decrement the first element to make it 0-indexed.
        for (int i = 1; i < counts.Length; i++)
        {
            counts[i] = counts[i] + counts[i - 1];
        }

        // Place the elements in sorted order in the output array.
```



```
for (int i = array.Length - 1; i >= 0; i--)
{
    sortedArray[counts[array[i] - minValue]--] = array[i];
}

// Display the sorted array.
Console.WriteLine("\n\nSorted array: ");
foreach (int num in sortedArray)
    Console.Write(num + " ");
Console.WriteLine("\n");
Console.ReadKey();
}
```

### | output |

Unsorted Array:  
3 4 -2 10 0 7 21 66 50 5 1 6

Sorted Array:  
-2 0 1 3 4 5 6 7 10 21 50 66



## Selection Sort

---

Selection Sort is a simple comparison-based sorting algorithm. It divides the list into two parts: a sorted part and an unsorted part. The algorithm iteratively selects the smallest (or largest, depending on the desired order) element from the unsorted part and moves it to the sorted part.

### How the Selection Sort Works

#### Initialization

---

Start with the entire array as the unsorted part, and an empty sorted part.

#### Iterative Process

---

At each step, select the smallest element in the unsorted part of the array. Swap this element with the first element of the unsorted part, placing it in the sorted part.

#### Repeat

---

Move the boundary between the sorted and unsorted parts one element to the right and repeat the process for the remaining unsorted elements.

#### End Condition

---

When the unsorted part contains only one element, the array is sorted.



## Time Complexity

### Best Case

$O(n^2)$  — Even if the array is already sorted, Selection Sort still goes through the whole array.

### Average Case

$O(n^2)$  — The algorithm does  $n-1$  comparisons for the first element,  $n-2$  for the second, and so on.

### Worst Case

$O(n^2)$  — Similar to the average case, since each pass involves comparing the remaining unsorted elements.

## Space Complexity

$O(1)$  — Selection Sort is an in-place sorting algorithm, meaning it requires only a constant amount of extra space for swapping elements.

## Real-World Use

- **Medal Distribution:** Sorting athlete performances to determine gold, silver, and bronze winners.
- **Gift Sorting:** Arranging presents under a Christmas tree by size or recipient priority.



## Example Code

---

### PowerShell

---

```
function Selection-Sort {
    param (
        [Parameter(Mandatory)]
        [array]$Array
    )

    # Display Unsorted Array
    Write-Host "Unsorted Array:"
    Display-Array-Elements -Array $Array

    # Perform Selection Sort
    for ($i = 0; $i -lt $Array.Length - 1; $i++) {
        $smallest = $i

        for ($index = $i + 1; $index -lt $Array.Length; $index++) {
            if ($Array[$index] -lt $Array[$smallest]) {
                $smallest = $index
            }
        }

        # Swap the smallest element with the current position.
        Swap-Elements -Array $Array -First $i -Second $smallest
    }

    # Display Sorted Array
    Write-Host "`nSorted Array:"
    Display-Array-Elements -Array $Array
}

function Swap-Elements {
    param (
        [array]$Array,
        [int]$First,
        [int]$Second
    )

    $temp = $Array[$First]
```



```
$Array[$First] = $Array[$Second]
$Array[$Second] = $temp
}

function Display-Array-Elements {
    param (
        [array]$Array
    )

    $Array -Join ' ' | Write-Host
}

Clear-Host

# Define and initialize the array.
$array = @(99, 7, 23, 6, 3, 9, 4, -1, 18, 2)

# Call the Selection-Sort function.
Selection-Sort -Array $array; Read-Host
```

### | output |

```
Unsorted Array:
99 7 23 6 3 9 4 -1 18 2

Sorted Array:
-1 2 3 4 6 7 9 18 23 99
```



## C#

---

```
using System;

namespace Selection_Sort
{
    class Program
    {
        static void Main(string[] args)
        {
            // Initialize the Selection_Sort class with the specific array.
            int[] customArray = new int[] { 99, 7, 23, 6, 3, 9, 4, -1, 18, 2
            };
            Selection_Sort selection = new Selection_Sort(customArray);
            selection.Sort();
            Console.ReadLine();
        }
    }

    class Selection_Sort
    {
        private int[] data;

        // Constructor takes in the predefined array.
        public Selection_Sort(int[] inputArray)
        {
            data = inputArray;
        }

        public void Sort()
        {
            // Display the unsorted array.
            Console.WriteLine("Unsorted Array:");
            display_array_elements();

            int smallest;
            // Perform Selection Sort
            for (int i = 0; i < data.Length - 1; i++)
            {
                smallest = i;

                for (int index = i + 1; index < data.Length; index++)
                {
                    if (data[index] < data[smallest])
                    {

```



```
        smallest = index;
    }
}

// Swap the smallest element with the current position.
Swap(i, smallest);
}

// Display the sorted array.
Console.WriteLine("\nSorted Array:");
display_array_elements();

}

// Swap the elements at the specified indices.
public void Swap(int first, int second)
{
    int temporary = data[first];
    data[first] = data[second];
    data[second] = temporary;
}

// Display the elements of the array.
public void display_array_elements()
{
    foreach (var element in data)
    {
        Console.Write(element + " ");
    }
    Console.WriteLine();
}
}
```

### | output |

```
Unsorted Array:
99 7 23 6 3 9 4 -1 18 2

Sorted Array:
-1 2 3 4 6 7 9 18 23 99
```



## Radix Sort

---

Radix Sort is a non-comparative integer sorting algorithm that sorts numbers by processing individual digits. It processes the digits from the least significant digit (LSD) to the most significant digit (MSD) or vice versa. Radix Sort works by distributing numbers into buckets according to each digit, then recombining them in a sorted order.

### How the Radix Sort Works

#### Initialization

---

Begin with the least significant digit (LSD) or the most significant digit (MSD), depending on the chosen method (LSD or MSD).

#### Digit-wise Sorting

---

Sort the elements based on the current digit (starting from the LSD or MSD). This can be done using a stable sorting algorithm (like counting sort or bucket sort).

#### Recombine

---

After sorting by each digit, recombine the numbers back into the array.

#### Repeat

---

Move to the next digit (either left or right) and repeat the sorting process until all digits are processed.

#### End Condition

---

The algorithm ends when all digits of the largest number have been processed.



## Time Complexity

### Best Case

$O(nk)$  — Where  $n$  is the number of elements, and  $k$  is the number of digits in the largest number.

### Average Case

$O(nk)$  — Similar to the best case.

### Worst Case

$O(nk)$  — Radix sort's performance does not degrade with input data.

## Space Complexity

$O(n)$  — Radix Sort requires extra space for sorting elements based on each digit.

## Real-World Use

- **Phone Numbers:** Organizing a directory of phone numbers numerically for easier lookups.
- **Sorting Serial Numbers:** Arranging large serial numbers in a manufacturing facility for inventory tracking.



## Example Code

---

### PowerShell

---

```
function Radix-Sort {
    param (
        [int[]]$arr
    )

    # Temporary array to hold elements while sorting.
    $tmp = New-Object int[] $arr.Length

    # Loop through each bit position from 31 down to 0.
    # An integer in PowerShell (like in C#) is 32 bits (4 bytes).
    # Sort the numbers from most significant bit (MSB)
    # to least significant bit (LSB).
    for ($shift = 31; $shift -ge 0; $shift--) {
        $j = 0

        # For each element in the array, we check if it should be placed
        # into the tmp array or remain in the original array based on the
        # current bit being checked.
        for ($i = 0; $i -lt $arr.Length; $i++) {
            # Determine if the bit at the current shift position is 1 or 0.
            $move = (($arr[$i] -ShL $shift) -ge 0)

            # If it's the most significant bit (shift == 0), we invert
            # the condition to ensure proper sorting of negative numbers at
            # the right place.
            if ($shift -eq 0 -and -not $move) {
                $arr[$i - $j] = $arr[$i] # Move element into original array.
            } else {
                $tmp[$j++] = $arr[$i] # Move element into temp array if 0.
            }
        }

        # Copy elements back from the temporary array to the original array.
        [Array]::Copy($tmp, 0, $arr, $arr.Length - $j, $j)
    }
}
```



```
$arr = @(99, 7, 23, 6, 3, 9, 4, -1, 18, 2)

Clear-Host

# Display Unsorted Array
Write-Host "Unsorted Array:"
$( $arr -Join ' ')

# Call the Radix-Sort function to sort the array.
Radix-Sort -Arr $arr

# Display Sorted Array
Write-Host "`nSorted Array:"
$( $arr -Join ' '); Read-Host
```

### | output |

```
Unsorted Array:
99 7 23 6 3 9 4 -1 18 2

Sorted Array:
99 7 23 6 3 9 4 -1 18 2
```



## C#

---

```
using System;

namespace Radix_Sort
{
    class Program
    {
        // Function to perform Radix Sort on an array.
        static void Sort(int[] arr)
        {
            int i, j;
            // Temporary array to hold elements while sorting.
            int[] tmp = new int[arr.Length];

            // Loop through each bit position from 31 down to 0.
            // This is because an integer in C# is 32 bits (4 bytes).
            // We sort the numbers from the most significant bit (MSB)
            // to the least significant bit (LSB).
            for (int shift = 31; shift > -1; --shift)
            {
                j = 0;

                // For each element in the array, we check if it should be
                // placed into the tmp array or remain in the original array
                // based on the current bit being checked.
                for (i = 0; i < arr.Length; ++i)
                {
                    // Determine if bit at current shift position is 1 or 0.
                    bool move = (arr[i] << shift) >= 0;

                    // If it's the most significant bit (shift == 0),
                    // we invert the condition to ensure.
                    // Proper sorting of negative numbers at the right place.
                    if (shift == 0 ? !move : move)
                        arr[i - j] = arr[i]; // Move element to original array
                    else
                        tmp[j++] = arr[i]; // Move element to temp array if 0.
                }

                // Copy elements from the temp array to the original array.
                Array.Copy(tmp, 0, arr, arr.Length - j, j);
            }
        }
    }
}
```



```
// Main method where the program execution starts.  
static void Main(string[] args)  
{  
    // Initialize an array of integers to be sorted.  
    int[] arr = new int[] { 1, 6, -4, 13, 0, 17, 20, 68, 50, 7 };  
  
    // Display unsorted array.  
    Console.WriteLine("Unsorted array:");  
    foreach (var item in arr)  
    {  
        Console.Write(item + " ");  
    }  
  
    // Call the Sort method to sort the array using Radix Sort.  
    Sort(arr);  
  
    // Display the sorted array after calling the Sort method.  
    Console.WriteLine("\n\nSorted array:");  
    foreach (var item in arr)  
    {  
        Console.Write(item + " ");  
    }  
    Console.WriteLine("\n");  
  
    // Wait for user input before closing the console.  
    Console.ReadLine();  
}  
}  
}
```

### | output |

```
Unsorted Array:  
1 6 -4 13 0 17 20 68 50 7  
  
Sorted Array:  
-4 0 1 6 7 13 17 20 50 68
```



## Bucket Sort

---

Bucket Sort is a distribution-based sorting algorithm that works by dividing the input array into several "buckets," sorting each bucket individually, and then combining the results. It is effective when the input data is uniformly distributed over a range.

### How the Bucket Sort Works

#### Create Buckets

---

Divide the input elements into several groups (buckets). The number of buckets typically depends on the range or size of the input data.

#### Distribute Elements

---

Place each element in the appropriate bucket based on a predetermined rule. Usually, this is done by mapping each element to a bucket based on its value or a specific range.

#### Sort Buckets

---

Sort each bucket individually. This can be done using any sorting algorithm (like insertion sort or bubble sort), depending on the bucket's size.

#### Combine Sorted Buckets

---

Concatenate the sorted buckets to form a sorted array.

#### End Condition

---

The algorithm completes when all buckets have been sorted and combined into a single array.



## Time Complexity

### Best Case

$O(n + k)$  — Where  $n$  is the number of elements and  $k$  is the number of buckets.

### Average Case

$O(n + k + k^2)$  — Where  $n$  is the number of elements and  $k$  is the number of buckets.

### Worst Case

$O(n^2)$  — If all elements are placed into one bucket and a slow sorting algorithm is used.

## Space Complexity

$O(n + k)$  — The algorithm requires extra space for the buckets and the sorted result.

## Real-World Use

- **Grades Distribution:** Sorting student grades into buckets (Ex: A, B, C) for quick assessment of performance distribution.
- **Rainfall Analysis:** Grouping daily rainfall data into buckets of ranges to analyze patterns over a month.



## Example Code

---

### PowerShell

---

```
function Bucket-Sort {
    param (
        [int[]]$arr
    )

    # Find the maximum and minimum values in the array.
    # These will determine the range for bucket creation.
    $max = $arr[0]
    $min = $arr[0]

    for ($i = 1; $i -lt $arr.Count; $i++) {
        if ($arr[$i] -gt $max) { $max = $arr[$i] }
        if ($arr[$i] -lt $min) { $min = $arr[$i] }
    }

    # Calculate the number of buckets required.
    # The number of buckets is based on the range of the numbers.
    $bucketCount = $max - $min + 1
    $holder = @()

    # Initialize each bucket as an empty array.
    for ($i = 0; $i -lt $bucketCount; $i++) {
        $holder += ,@()
    }

    # Distribute the elements into the corresponding buckets.
    # Each element is placed in the bucket based on its value
    # relative to the minimum.
    for ($i = 0; $i -lt $arr.Count; $i++) {
        $bucketIndex = $arr[$i] - $min
        $holder[$bucketIndex] += $arr[$i]
    }

    # Combine all buckets into a single sorted array.
    $sortedArray = @()
    for ($i = 0; $i -lt $holder.Count; $i++) {
        if ($holder[$i].Count -gt 0) {
```



```
# Append all elements from the current bucket to the sorted array.  
$sortedArray += $holder[$i]  
}  
}  
  
# Return the sorted array to the caller.  
return $sortedArray  
}  
  
# Test the Bucket-Sort function.  
  
Clear-Host  
  
# Define the unsorted array.  
$arr = @(99, 7, 23, 6, 3, 9, 4, -1, 18, 2)  
  
# Display Unsorted Array  
Write-Host "Unsorted Array:"  
Write-Host ($arr -Join ' ')  
  
# Call the Bucket-Sort function to sort the array.  
$arr = Bucket-Sort -Arr $arr  
  
# Display Sorted Array  
Write-Host "`nSorted Array:"  
Write-Host ($arr -Join ' '); Read-Host
```

## | output |

```
Unsorted Array:  
99 7 23 6 3 9 4 -1 18 2  
  
Sorted Array:  
-1 2 3 4 6 7 9 18 23 99
```



## C#

---

```
using System;
using System.Collections.Generic;

namespace BucketSort
{
    public class BucketSort
    {
        static void Main(string[] args)
        {
            // Initialize an unsorted array of integers.
            int[] x = new int[] { 98, 96, 89, 84, 79, 74, 71, 66, 61, 54, 50,
            44, 39, 36, 29, 24, 19 };

            // Display Unsorted Array
            Console.WriteLine("Unsorted Array:");
            // Join elements with commas for cleaner output.
            Console.WriteLine(string.Join(", ", x));

            // Call the sorting function and store the result in 'sorted'.
            List<int> sorted = Sort(x);

            // Display Sorted Array
            Console.WriteLine("\nSorted Array:");

            // Join elements with commas for cleaner output.
            Console.WriteLine(string.Join(", ", sorted));

            // Wait for the user to press a key before closing the program.
            Console.Read();
        }

        // Function to perform bucket sort on an array of integers.
        public static List<int> Sort(params int[] x)
        {
            // Initialize an empty list to store the sorted result.
            List<int> result = new List<int>();

            // Set the number of buckets
            int numOfBuckets = 10;

            // Create an array of lists (buckets) to hold elements.
            List<int>[] buckets = new List<int>[numOfBuckets];
```



```
for (int i = 0; i < numOfBuckets; i++)
    // Initialize each bucket as a new list.
    buckets[i] = new List<int>();

// Distribute elements into buckets based on their value.
for (int i = 0; i < x.Length; i++)
{
    // Determine which bucket the element belongs to.
    int bucketChoice = (x[i] / numOfBuckets);
    // Add the element to the appropriate bucket.
    buckets[bucketChoice].Add(x[i]);
}

// For each bucket, sort elements and add them to the result list.
for (int i = 0; i < numOfBuckets; i++)
{
    // Sort the elements of the bucket using bubble sort.
    int[] temp = BubbleSortList(buckets[i]);
    // Add sorted elements of current bucket to the result list.
    result.AddRange(temp);
}

// Return the sorted result.
return result;
}

// Function to sort a list of integers using bubble sort.
public static int[] BubbleSortList(List<int> input)
{
    // Perform bubble sort.
    for (int i = 0; i < input.Count; i++)
    {
        // Inner loop for comparing adjacent elements.
        // Optimization to avoid unnecessary comparisons.
        for (int j = 0; j < input.Count - i - 1; j++)
        {
            // If elements are out of order, swap them.
            if (input[j] > input[j + 1])
            {
                int temp = input[j];
                input[j] = input[j + 1];
                input[j + 1] = temp;
            }
        }
    }
    // Return the sorted list as an array.
    return input.ToArray();
}
```



```
    }  
}  
}
```

### | output |

Unsorted Array:

98, 96, 89, 84, 79, 74, 71, 66, 61, 54, 50, 44, 39, 36, 29, 24, 19

Sorted Array:

19, 24, 29, 36, 39, 44, 50, 54, 61, 66, 71, 74, 79, 84, 89, 96, 98



## Permutation Sort

---

Permutation Sort is a sorting algorithm based on generating all permutations of the given input and then selecting the sorted one. While conceptually simple, it's impractical for large datasets due to its extremely high time complexity.

### How the Permutation Sort Works

#### Find the Range

---

Identify the minimum and maximum values in the input array to determine the range of values.

#### Initialize Count Array

---

Create a count array with a size equal to the range of values, where each index represents the frequency of a particular element in the input array.

#### Count Frequencies

---

Iterate through the input array and increment the corresponding index in the count array based on each element's value.

#### Cumulative Count

---

Modify the count array by converting it to a cumulative count, where each index represents the position of the element in the sorted array.

#### Place Elements in Sorted Order

---

Using the cumulative count array, place the elements from the input array into their correct positions in the output array.



## End Condition

---

The algorithm completes when all elements are placed in the output array, resulting in a sorted array.

## Time Complexity

### Best Case

---

$O(n!)$  — Where  $n$  is the number of elements. The best case occurs when the input array is already sorted, and the algorithm will still need to check all permutations.

### Average Case

---

$O(n!)$  — Where  $n$  is the number of elements. The average case also involves checking all permutations.

### Worst Case

---

$O(n!)$  — Where  $n$  is the number of elements. The worst case happens when the input is in reverse order, and the algorithm has to generate and check all  $n!$  permutations.

## Space Complexity

$O(n)$  — The algorithm requires space to store the list and the recursion stack.

## Real-World Use

- **Jigsaw Puzzle Arrangement:** Testing different permutations of puzzle pieces to find the correct arrangement.
- **Solving Anagrams:** Rearranging letters in a word to generate all possible valid anagrams.



## Example Code

---

### PowerShell

---

```
function Swap {
    param (
        [ref]$a,
        [ref]$b
    )

    # If both characters are the same, no need to swap.
    if ($a.Value -eq $b.Value) { return }

    # XOR-based swapping (no temporary variable needed).
    # Write-Host "Swapping: $($a.Value) with $($b.Value)"
    $a.Value = [char](([int][char]$a.Value -bxor [int][char]$b.Value))
    $b.Value = [char](([int][char]$a.Value -bxor [int][char]$b.Value))
    $a.Value = [char](([int][char]$a.Value -bxor [int][char]$b.Value))

    # Display the array state after the swap.
    # Write-Host "Array after swap: $([string]::Join('', $list))"
}

function Get-Per {
    param (
        [char[]]$list
    )

    $x = $list.Length - 1
    Get-PerRecursive -List $list -k 0 -m $x
}

function Get-PerRecursive {
    param (
        [char[]]$list,
        [int]$k,
        [int]$m
    )

    # If k == m, display the current permutation.
    if ($k -eq $m) {
```



```
# Write-Host "Permutation found: $([string]::Join('', $list))"
} else {
    # Generate all permutations for the current level.
    for ($i = $k; $i -le $m; $i++) {
        # Swap the current index with the iteration index.
        # Write-Host "Processing: $([string]::Join('', $list))"
        Swap ([ref]$list[$k]) ([ref]$list[$i])

        # Recursively permute the remaining elements.
        Get-PerRecursive -List $list -k ($k + 1) -m $m

        # Backtrack: Restore the array to its previous state.
        # Write-Host "Backtracking: $([string]::Join('', $list))"
        Swap ([ref]$list[$k]) ([ref]$list[$i]) # Swap back
    }
}
}

Clear-Host

# Input string (unsorted array of characters).
$str = "BAC"
$arr = $str.ToCharArray()

# Display Unsorted Array
Write-Host "Unsorted Array:"
Write-Host "$str"
Write-Host ""

# Display the processing steps.
# Write-Host "Processing Permutations:"
# Get-Per -List $arr

# Sort the array.
$arr = $arr | Sort-Object

# Display Sorted Array
Write-Host "`nSorted Array:"
Write-Host ([string]::Join("", $arr))
Read-Host
```



## | output |

Unsorted Array:  
BAC

Sorted Array:  
ABC



## C#

---

```
using System;

class Program
{
    static void Main()
    {
        // Define the input string (unsorted array of characters).
        string str = "BAC";
        char[] arr = str.ToCharArray();

        // Display Unsorted Array
        Console.WriteLine("Unsorted Array:");
        Console.WriteLine(new string(arr));
        Console.WriteLine();

        // Process and display permutations.
        Console.WriteLine("Processing Permutations:");
        GetPer(arr);

        // Display Sorted Array
        Array.Sort(arr);
        Console.WriteLine("\nSorted Array:");
        Console.WriteLine(new string(arr));

        // Wait for user input before exiting.
        Console.Read();
    }

    // Swap two characters in place.
    private static void Swap(ref char a, ref char b)
    {
        // If both characters are the same, no need to swap.
        if (a == b) return;

        // XOR-based swapping (no temporary variable needed).
        a ^= b;
        b ^= a;
        a ^= b;
    }

    // Entry point for generating all permutations of the character array.
    public static void GetPer(char[] list)
    {
```



```
        int x = list.Length - 1;
        GetPer(list, 0, x);
    }

    // Recursive method to generate permutations.
    private static void GetPer(char[] list, int k, int m)
    {
        // If k == m, we've generated one permutation.
        if (k == m)
        {
            // Display current permutation.
            Console.WriteLine(new string(list));
        }
        else
        {
            // Generate all permutations for the current level.
            for (int i = k; i <= m; i++)
            {
                // Swap the current index with the iteration index.
                Swap(ref list[k], ref list[i]);

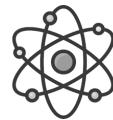
                // Recursively permute the remaining elements.
                GetPer(list, k + 1, m);

                // Backtrack: Restore the array to its previous state.
                Swap(ref list[k], ref list[i]);
            }
        }
    }
}
```

## | output |

Unsorted Array:  
BAC

Sorted Array:  
ABC



## UNIFIED SOLUTIONS



### 15. Integration

---

Unified Solutions is about bringing it all together—full integration—with everything you've learned about data structures, algorithms, scripting, programming, and implementing real-world solutions. These next few sections are about the culmination of your journey thus far. By now, you've explored multiple scripting languages and tackled C#. You've learned to use code to complete common tasks, solve problems, and automate processes. You should be proud of yourself if you're here and ready to move forward. You have accomplished a lot. In Unified Solutions, we take things to the next level by combining all those skills into cohesive, self-contained solutions that are not only functional but also polished and professional.

This isn't just about compiling scripts or packaging resources; it's about the power of integration. Bringing all these concepts together demonstrates how coding isn't just a collection of isolated skills but a framework for solving everyday challenges in creative and efficient ways. Whether it's optimizing workflows, managing complex systems, or automating repetitive tasks, Unified Solutions highlights the transformative potential of your knowledge.

Why does this matter? Because mastering Unified Solutions unlocks the ability to bridge gaps between various disciplines and turn them into cohesive outcomes. It's not just about creating something that works—it's about creating something that works well, stands up to business demands, and improves the lives of its users.

Every bit of knowledge you've gained so far—from designing efficient scripts to crafting compiled code—has been building toward this point. Unified Solutions is where you take control of all these moving parts, orchestrating them to work in unison. It's an exercise in creativity, problem-solving, and technical skill that shows just how far you've come.



## What will be covered?

### Section 16: Flowcharts for Code

- **Code Design:** Map out the logic and flow of your scripts, or any code, to visualize the steps, conditions, and decision points involved. This approach ensures that your script is well-organized and easy to follow, whether it's processing data, checking system status, or interacting with external resources.

### Section 17: Compilation & Packaging

- **Compile:** Learn how to compile your Batch and PowerShell scripts into single, executable packages. You can customize things like company name, icon, and hidden status. Creates a self-contained, portable file.
- **Package:** Learn how to package your Batch and PowerShell scripts with resource files into single, executable packages. By bundling external files, configuration settings, and dependencies, you ensure that your scripts run seamlessly across various systems, improving reliability and ease of use.

### Section 18: Deployment

- **Bringing it Full Circle:** How to leverage your self-contained packages to solve real-world problems. Whether it's deploying scripted solutions to hundreds of endpoints or creating tools that your organization can rely on, this section brings all your skills together in a meaningful way.



## 16. Flowcharts for Code

---

In IT and system administration, clarity and organization are essential. Whether troubleshooting code or managing complex infrastructures, flowcharts are invaluable tools for mapping processes and strategies. They simplify workflows, making it easier to identify bottlenecks, optimize procedures, and communicate effectively with both technical and non-technical stakeholders. Flowcharts provide a high-level overview, enabling quicker decision-making and smoother coordination.

This section will focus on the use of flowcharts for your code, a powerful method for streamlining workflows and enhancing system management.

Diagramming code logic helps developers and administrators visualize decision trees, loops, and potential errors, accelerating debugging, improving readability, and fostering collaboration among team members. By creating flowcharts of code logic, developers can uncover inefficiencies, redundancies, and potential issues, leading to more streamlined and efficient code. This approach serves as a clear and intuitive guide for presenting the logic to others or revisiting the code at a later time. In essence, flowcharting code transforms complex algorithms into a visual format that is easier to understand, optimize, and maintain.

I will present three code design examples using flowcharts, each demonstrating increasing complexity. Keep in mind, as code segments become more complex, flowcharts become increasingly valuable for understanding and organizing the logic. Entire books have been written about flowcharts, but this should provide a solid starting point for you.

For more about flowcharts beyond this section, see:

<https://github.com/mrnettek/Atomics/blob/main/Flowchart.txt>

Let's take a look at our examples.

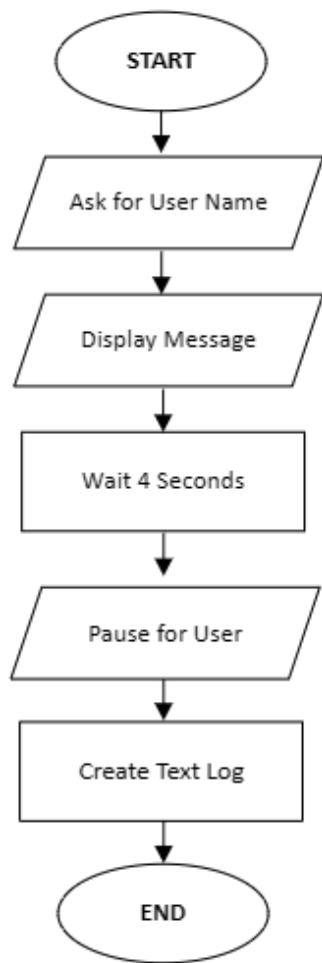


## Flowchart Example 1

---

\* All examples created in Visio.

Without me providing any information, try to guess how the code would work.



If this looks familiar, that's because it is—the flowchart mirrors the basic structure of our Batch script basic design. If you found it straightforward to understand, chances are it will be just as clear for your manager or other project stakeholders.



## Assessment

**Start (Oval):** Indicate the beginning of the script.

**Input (Parallelogram):** Ask the user, "What is your name?"

**Output (Parallelogram):** Display the message "Hello, [Name]! Have a great day!"

**Process (Rectangle):** Wait for 4 seconds using TIMEOUT.

**Input (Parallelogram):** Pause and wait for user input.

**Process (Rectangle):** Log the success message to log.txt.

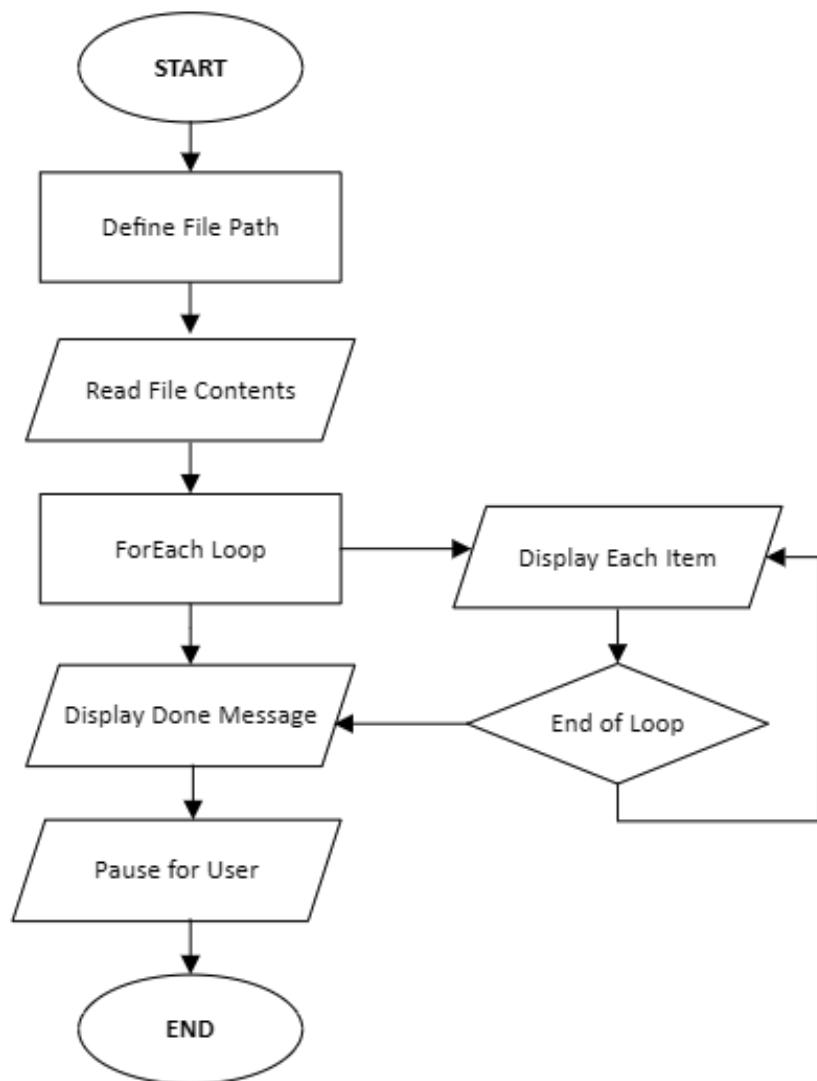
**End (Oval):** Terminate the script.



## Flowchart Example 2

---

In this example, we diagram a PowerShell script that loads the contents of a text file into an array, then iterates through each line, displaying console lines using Write-Host.





## Assessment

**Start (Oval):** Represents the start of the script.

**Process (Rectangle):** Define the file path (\$filePath = 'C:\PowerShell\File.txt').

**Input (Parallelogram):** Read the file content into an array (\$Array = Get-Content -Path "\$filePath").

**Process (Rectangle):** Initialize a loop with ForEach(\$Item in \$Array) to iterate through the array.

**Output (Parallelogram):** Inside the loop, output the current item (Write-Host \$Item).

**Loop back to ForEach:** After processing an item, loop back to process the next item in the array.

**End of Loop (Diamond):** Check if there are more items in the array to process.

- **False, not at end of loop:** Continue the loop to process the next item.
- **True, at end of loop:** Proceed to Done..

**Output (Parallelogram):** Display "Done!" after completing the loop.

**Input (Parallelogram):** Prompt the user to press a key to continue.

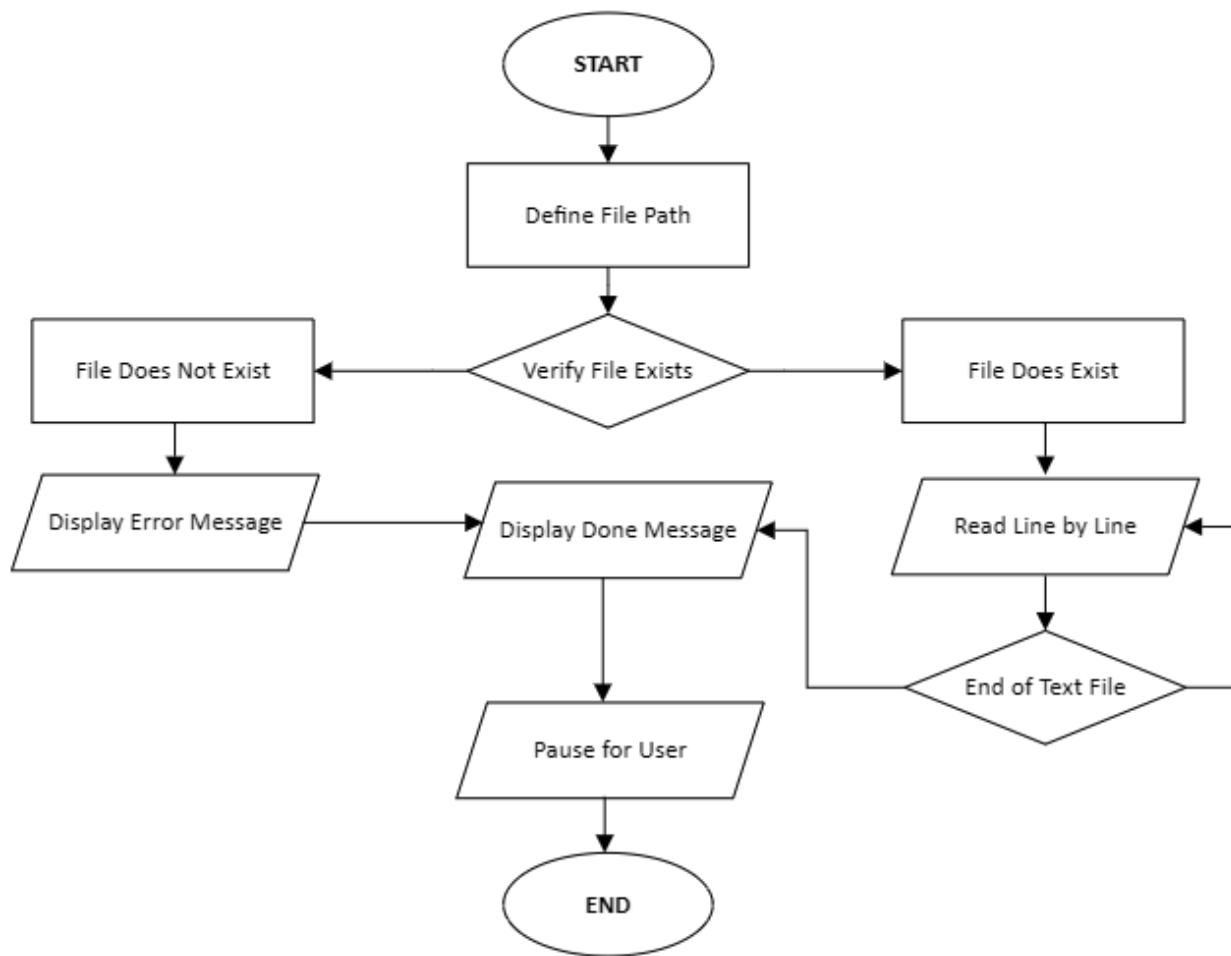
**End (Oval):** Represents the end of the script.



## Flowchart Example 3

---

In this example, we diagram the PowerShell script that reads a text file line by line. I did leave out the counter just for the sake of brevity.





## Assessment

**Start (Oval):** Represents the start of the script.

**Process (Rectangle):** Define the file path.

**Decision (Diamond):** Check if the file exists (Test-Path).

- **True, Path Exists (Rectangle):**

1. **Process (Rectangle):** Read the file's content line by line.
2. **Decision (Diamond):** Check if the end of the file has been reached:
  - **False, not at end of file:**
    - Go back to Read Line by Line.
  - **True, at of file:**
    - Go to the Done message.

- **False, Path Does Not Exist (Rectangle):**

1. **Output (Parallelogram):** Display "The file does not exist."
2. Go to the Done message.

**Output (Parallelogram):** Display "Done!" after completing the file processing or if the file does not exist.

**Input (Parallelogram):** Prompt the user to press a key to continue.

**End (Oval):** Represents the end of the script.



## Flowchart Shapes

---

Here's a table of common flowchart shapes and their descriptions.

Shape	Description	Purpose
<b>Oval (Terminator)</b>	Ellipse shape	Marks the start or end of a process.
<b>Rectangle (Process)</b>	A simple rectangle	Represents a process or action step (Ex: calculation, operation).
<b>Parallelogram (Input/Output)</b>	A parallelogram	Used to show input or output operations (Ex: user input, displaying data).
<b>Diamond (Decision)</b>	A diamond shape	Indicates a decision point, typically with Yes/No or True/False outcomes.
<b>Arrow (Flowline)</b>	A simple arrow	Shows the direction of flow from one step to another in the process.
<b>Circle (Connector)</b>	A small circle	Used to connect different parts of the flowchart, especially for large or complex flowcharts.
<b>Document</b>	A rectangle with a wavy bottom	Represents a document or report generated in the process.
<b>Predefined Process (Subroutine)</b>	A rectangle with double vertical lines on the sides	Indicates a predefined process or subroutine, usually referring to a separate flowchart.
<b>Data (Stored Data)</b>	A rectangle with a slanted bottom	Denotes data storage, such as a file or database.
<b>Manual Operation</b>	A trapezoid	Represents a step that is manually performed, often in contrast to automated operations.



## 17. Compilation & Packaging

---

Compiling scripts into a single, portable file is one of those "aha" moments for anyone using automation tools. Why? Because it just makes life easier. Imagine not having to chase down missing files or explain why your carefully crafted script won't run without some obscure configuration. By bundling everything into one neat executable package—scripts, configurations, and any other necessary files—you not only avoid headaches but also make deployment a breeze. Plus, your source code stays safe from prying eyes, which is a win for security and peace of mind.

If you're working with deployment tools like SCCM, Intune, or Ivanti, this practice really shines. Having one self-contained executable means you can push it out to endpoints without worrying about scattered resources. It's efficient, it's reliable, and it's the kind of thing that makes you look like the rockstar admin who has it all figured out.

But the benefits don't stop there (but, wait, there's even more). Compiling scripts ensures that your tools are portable, which is especially useful when you're managing diverse environments or sending your work to colleagues who may not have the same setups. It's also a massive time-saver. Instead of troubleshooting why something didn't work, you can confidently provide a single executable that just works. And when you're dealing with sensitive information or intellectual property, having a compiled file means your scripts are protected from unauthorized edits or exposure.

There's also the professional edge it gives you. A polished, all-in-one executable screams expertise and reliability. Imagine sending your users a tool that doesn't just function but feels seamless and user-friendly. You're not just solving a problem; you're leaving a lasting impression.

Once you master the art of compiling and packaging scripts, you'll find yourself wondering how you ever managed without it, and you'll see its value in everything from day-to-day troubleshooting to large-scale deployments. Any script you can create will become elevated with the proper compilation and packaging. For those who have no interest in higher level programming languages, compiling your scripts with resources is as close as you'll get to the benefits of compiled code in languages like C#.



## Benefits of Compiling Scripts & Packaging Resources

- **Portability:** Consolidate scripts and resources into a single file, making them easy to share and distribute.
- **Error Reduction:** Prevent issues caused by missing or misplaced resource files.
- **Streamlined Deployment:** Simplify deployment processes with tools like SCCM, Intune, or Ivanti.
- **Improved User Experience:** Reduce the need for end-users to handle multiple files or complex setups.
- **Version Control:** Ensure consistent execution by embedding all dependencies into a single package.
- **Convenience:** Automate the extraction and execution of resources during runtime.
- **Enhanced Security:** Protect sensitive data by embedding it securely within the executable.
- **Time Savings:** Eliminate the need for manual setup by end-users, speeding up onboarding and deployment.
- **Professional Presentation:** Create polished, user-friendly solutions that reflect expertise.
- **Cross-Platform Compatibility:** Ensure consistent behavior across various systems and environments.
- **Reduced Dependency Management:** Minimize reliance on external libraries or files, reducing support calls and troubleshooting.

Let's go through the script compilation and packaging steps, and then we'll review some screenshots from some tools I use.



## Compiling Batch Scripts

---

### Common Tools

- ExeScript
- Bat2Exe Converter
- Advanced BAT to EXE Converter
- WinRAR (SFX Creation)

### Step-by-Step: Without Resource Files

#### 1. Prepare the Script

1. Save your batch script as sequence.cmd.

#### 2. Use a Batch Compiler

1. Open your chosen tool  
(Example: ExeScript).
2. Load the script (sequence.cmd).
3. Choose the desired settings  
(examples: add icon, hide the console window).
4. Compile the script to an EXE file.

#### 3. Test the Output and Processing

1. Run the compiled sequence.exe to ensure it executes correctly.
2. Test in the System account.



## Step-by-Step: With Resource Files

### 1. Prepare Resources

1. Save the files (config.txt, setup.cer, setup.msi) in the same folder as sequence.cmd.

### 2. Embed Resources

1. Use a tool like ExeScript or Advanced BAT to EXE Converter.
2. Specify the files to embed during the compilation process.

### 3. Compile

1. Follow the tool's instructions to generate the EXE.

### 4. Embed Resources

1. Use a tool like ExeScript or Advanced BAT to EXE Converter.

### 5. Test the Output and Processing

1. Verify that the resources are extracted and the script functions correctly.
2. Test in the System account.



## Compiling PowerShell Scripts

---

### Common Tools

- PS2EXE
- WinRAR SFX Creation (Easy, self-extracting EXE)
- PowerGU Script Editor (Old, but still good)

### Step-by-Step: Without Resource Files

#### 1. Prepare the Script

1. Save your PowerShell script as sequence.ps1.

#### 2. Use PS2EXE.ps1

1. Install the PS2EXE module (Install-Module PS2EXE if not already installed).
2. Run the following command to compile:

```
ps2exe.ps1 -inputFile sequence.ps1 -outputFile sequence.exe
```

3. Customize options (Examples: icon, console visibility) as needed.

#### 3. Test the Output and Processing

1. Execute the compiled sequence.exe to ensure proper functionality.
2. Test in the System account.

### NOTE

---

<https://github.com/MScholtes/PS2EXE>



## Step-by-Step: With Resource Files

### 1. Prepare Resources

1. Save config.txt, setup.cer, and setup.msi in the same folder as sequence.ps1.

### 2. Embed Resources

1. Save config.txt, setup.cer, and setup.msi in the same folder as sequence.ps1. I recommend converting these to Base64, and then the Base64 string is what you put into the sequence.ps1. Once you do it once, you won't know how you lived without knowing about this.

This is how you create the Base64 files. Add the paths to files you want to convert to Base64 (in orange) and run the script.

```
param (
    # Add files here. Specify comma-separated list to convert.
    [string]$FilePaths =
    "C:\PowerShell\config.txt,C:\PowerShell\setup.cer,C:\PowerShel
    l\setup.msi"
)

# Convert the comma-separated list to an array.
$ filePathsArray = $FilePaths.Split(',')

# Loop through each file in the list.
foreach ($filePath in $filePathsArray) {

    # Check if the file exists.
    if (-not (Test-Path -Path $filePath)) {
        Write-Error "The specified file '$filePath' does not
exist."
        continue
    }

    try {
        # Read the file and encode it to Base64.
        $fileContent = Get-Content -Path $filePath -Encoding Byte
        $base64Content = [Convert]::ToBase64String($fileContent)

        # Create an output file name.B64.txt.
        $outputFile = "$($filePath).B64.txt"
    }
}
```



```
# Write the Base64 content to the output file.  
Set-Content -Path $outputFile -Value $base64Content -Force  
  
Write-Host "Base64 encoding for '$filePath' completed  
successfully."  
  
Write-Host "Encoded file saved to: $outputFile"  
} catch {  
    Write-Error "An error occurred during the encoding  
process for '$filePath': $_"  
}  
}
```

2. Now take the contents of each generated text file and copy that long string of data to the relative string in the sequence.ps1. See script below.
3. Use PS2EXE to include the resources during compilation, assuming the Base64 has been completed. Otherwise you may choose another tool for packaging.
4. Include commands in sequence.ps1 to extract embedded files. Put the extract commands before any setup or task automation which may require your resource files. The rest of your code will follow underneath the extraction.

```
# Embedded Base64 content for files.  
$setupCerBase64 = @"  
 <Replace this with Base64 content of setup.cer>  
"@  
  
$setupMsiBase64 = @"  
 <Replace this with Base64 content of setup.msi>  
@"  
  
$setupConfigBase64 = @"  
 <Replace this with Base64 content of setup.msi>  
@"  
  
# Function to decode Base64 content and save as a file.  
function Write-EmbeddedFile {  
    param (  
        [string]$Base64Content,  
        [string]$OutputFile  
    )  
    $Bytes = [Convert]::FromBase64String($Base64Content)  
    [System.IO.File]::WriteAllBytes($OutputFile, $Bytes)  
}
```



```
# Define the output path
$outputPath = "$env:TEMP\ExtractedFiles"
New-Item -Path $outputPath -ItemType Directory -Force

# Write embedded files to disk.
Write-EmbeddedFile -Base64Content $setupCerBase64 -OutputFile
"$outputPath\setup.cer"

Write-EmbeddedFile -Base64Content $setupMsiBase64 -OutputFile
"$outputPath\setup.msi"

Write-EmbeddedFile -Base64Content $setupConfigBase64
-OutputFile "$outputPath\config.txt"

# Install the MSI silently using msieexec.
Start-Process MSIEEXEC -ArgumentList '/I',
"$outputPath\setup.msi", '/QN', '/NORESTART' -NoNewWindow
-Wait

# Define the registry path and value details.
$regPath = "SOFTWARE\TEST"
$regKey = "myKey"
$regValue = "1"

# Open the registry hive for 64-bit view.
$regHive =
[Microsoft.Win32.RegistryKey]::OpenBaseKey([Microsoft.Win32.R
egistryHive]::LocalMachine,
[Microsoft.Win32.RegistryView]::Registry64)
# Create or open the Registry key.
$regSubKey = $regHive.CreateSubKey($regPath)

# Set the value.
$regSubKey.SetValue($regKey, $regValue,
[Microsoft.Win32.RegistryValueKind]::String)

# Close the key to release resources.
$regSubKey.Close()

$certPath = "$outputPath\setup.cer"

$store = New-Object
System.Security.Cryptography.X509Certificates.X509Store("Root",
", "LocalMachine")

$store.Open("ReadWrite")

$cert = New-Object
```



```
System.Security.Cryptography.X509Certificates.X509Certificate  
2($certPath)  
  
# Add our cert to the specified store.  
$store.Add($cert)  
  
$store.Close()
```

### 3. Compile to Executable

1. Generate the executable, ensuring all embedded resources are packaged. At a PowerShell prompt:

```
ps2exe.ps1 -inputFile sequence.ps1 -outputFile sequence.exe
```

Or

```
ps2exe.ps1 -inputFile sequence.ps1 -outputFile sequence.exe  
-noConsole -iconFile "C:\PowerShell\icon.ico"
```

Or

```
ps2exe.ps1 -inputFile sequence.ps1 -outputFile sequence.exe  
-noConsole -iconFile "C:\PowerShell\icon.ico" -company  
"Company Name" -product "App Name" -Version "1.0.0"  
-description "App Description"
```

### 4. Test the Output and Processing

1. Confirm the EXE extracts resources and runs the script as expected.
2. Test in the System account.



## Alternative Method

---

I'd be doing you a disservice if I didn't share my own approach to compiling PowerShell scripts and resource files. This method works seamlessly with most Batch compilers that also support packaging resource files alongside the .cmd file. I prefer this approach because it simplifies the process and minimizes the number of steps and tools needed to create polished, ready-to-use solutions. Here are the text-based steps, but we'll also take a look at these in the visual guides.

### Batch Compiler for PowerShell. Say, what?

1. Add the PowerShell sequence.ps1 as a resource file in your Batch compiler and any other resource files your PowerShell script may require.
2. In your Batch *sequence.cmd*, add this code referencing your *sequence.ps1*:

```
CD "%~dp0"  
PowerShell -File .\sequence.ps1
```

3. Compile to Executable.
4. Test.
5. Test in the System account.

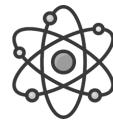
### NOTE

---

Using a local UNC (admin share) guarantees the correct architecture's bit version.

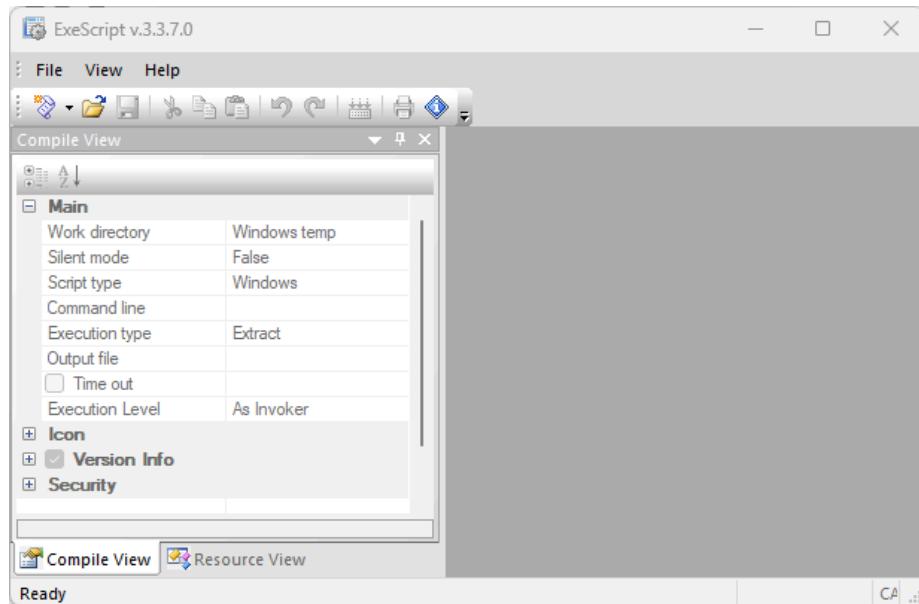
So, you could compile your scripts using a 32 bit app, but the PowerShell script would still execute as 64 bit.

```
\%COMPUTERNAME%\C$\Windows\system32\WindowsPowerShell\v1.0\PowerShell.exe  
-NoProfile -WindowStyle Hidden -ExecutionPolicy Bypass -File .\sequence.ps1
```

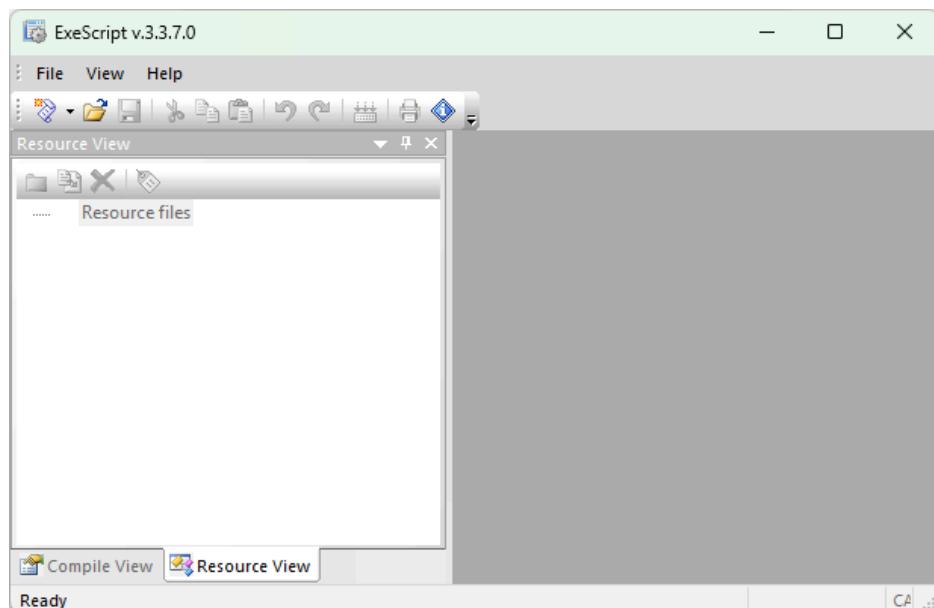


## Visual Guide to Compiling a Batch Script

This is our Batch compiler, ExeScript. No properties are configured in the left pane yet.



Also, no resource files have been added. If we look to the right pane, there is no code, so we have yet to load a script.





**Step 1** | We load a script, or create a new script using the code editor. The script we load will silently install setup.msi, import an app cert, and add reg keys to HKLM and HKEY\_USERS. This code is from a previous example, if you remember.

```
BAT script.cmd
1  @rem ----- ExeScript Options Begin -----
2  @rem ScriptType: console,silent,invoker
3  @rem DestDirectory: C:\Batch
4  @rem Icon: C:\Windows\System32\shell32.dll,314
5  @rem File: C:\Batch\config.txt
6  @rem File: C:\Batch\setup.cer
7  @rem File: C:\Batch\setup.msi
8  @rem OutputFile: C:\Batch\Scripted-Installer.exe
9  @rem CompanyName: MrNetTek
10 @rem FileDescription: Scripted Setup
11 @remFileVersion: 1.0.0.1
12 @rem LegalCopyright: MrNetTek
13 @rem ProductName: Scripted Installer
14 @rem ProductVersion: 1.0.0.1
15 @rem ----- ExeScript Options End -----
16 SETLOCAL EnableDelayedExpansion
17 :: Install the software
18 MSIEXEC /I "C:\Batch\setup.msi" /QN /NORESTART
19
20 :: Import a certificate into the user's certificate store
21 ECHO Importing certificate... & TIMEOUT /T 2 >nul
22 CERTUTIL -ADDSTORE "Root" "C:\Batch\setup.cer"
23 ECHO.
24
25 :: Add registry entries for the application.
26 ECHO Adding registry keys... & TIMEOUT /T 2 >nul
27 ECHO Adding hklm key to HKEY_LOCAL_MACHINE\Software\TEST
28 REG ADD "HKEY_LOCAL_MACHINE\Software\TEST" /V "myKey" /D
29
30 :: Return User SID
31 FOR /F "tokens=3" %%A IN ('REG QUERY      "HKLM\SOFTWARE\M
32 | SET "uSID=%%A"
33
34 ECHO Adding hku key to HKEY_USERS!\uSID!\Software\TEST..
35 REG ADD "HKEY_USERS!\uSID!\Software\TEST" /V "myKey" /D 1
36 )
37
38 ENDLOCAL & ECHO. & ECHO Done! & ECHO. & PAUSE
39
40
41
42 EXIT /B 0
```

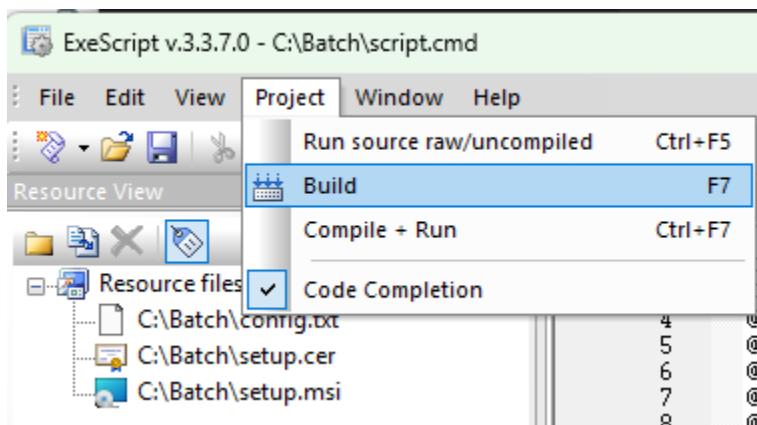


**Step 2** | We configure the script executable properties, such as silent mode, a custom icon, and company information.

**Step 3** | We add any resources. In this example, config.txt, setup.cer, and setup.msi are our resource files.



**Step 4** | We select the Build option from the menu. This will package the script.cmd and resources files into a single, portable executable file. Magical.



**Step 5** | Let's find the Scripted-Installer.exe file that has appeared in the C:\Batch folder. Since we make changes to the registry and program files, run as administrator.

Upon running our executable on a test machine, the setup.msi runs silently to complete its installation, the certificate is imported, and two new registry keys have been created. Everything has worked as intended. Success won't always be the case, but that's why we test. You will often spend more time testing than actually developing solutions.

Name	Type	Size
config.txt	Text Document	1 KB
script.cmd	Windows Command Script	1 KB
setup.cer	Security Certificate	1 KB
setup.msi	Windows Installer Package	67,308 KB
Scripted-Installer.exe	Application	67,446 KB

**Wait a minute.** You may be asking, "What was that config.txt file?" The config.txt was extra, just to show you other files could be included. Config files are very common, and come in different shapes and sizes. A simple config file could be used to apply advanced settings during an install or uninstall, or assist with any other form of desktop management task (you'll see .mst, .ini, .txt, .reg, and .dat files used for applying or importing settings). But for now, just try to understand how each step works. It's not so mysterious once you see it in action. I'm proud of you for making it this far.



## Visual Guide to Compiling a PowerShell Script

### Example 1 of 2

Using PS2EXE, we will create a single, self-contained executable file. We will add screenshots to the PowerShell example we previously reviewed.

**Step 1 |** Save config.txt, setup.cer, and setup.msi in the same folder as sequence.ps1. I recommend converting the resource files to Base64, and then the Base64 string is what you put into the sequence.ps1. Once you do it once, you won't know how you ever lived without it.

Name	Type	Size
test	File folder	
config.txt	Text Document	1 KB
ps2exe.ps1	Windows PowerShell Script	117 KB
sequence.ps1	Windows PowerShell Script	500 KB
setup.cer	Security Certificate	1 KB
setup.msi	Windows Installer Package	374 KB

**Step 2 |** This is how you create Base64 files. Add the paths to files you want to convert to Base64 (in orange) and run the script. We do this to embed the files as code into our sequence.ps1 script. Then, there is no need to package the resource files.

```
param (
# Add files here. Specify a comma-separated list to convert.
[string]$FilePaths =
"C:\PowerShell\config.txt,C:\PowerShell\setup.cer,C:\PowerShell\setup.msi"
)

# Convert the comma-separated list to an array.
$ filePathsArray = $FilePaths.Split(',')

# Loop through each file in the list.
foreach ($filePath in $filePathsArray) {
```



```
# Check if the file exists.
if (-not (Test-Path -Path $filePath)) {
    Write-Error "The specified file '$filePath' does not exist."
    continue
}
try {
    # Read the file and encode it to Base64.
    $fileContent = Get-Content -Path $filePath -Encoding Byte
    $base64Content = [Convert]::ToBase64String($fileContent)

    # Create an output file name.B64.txt.
    $outputFile = "$( $filePath ) .B64.txt"

    # Write the Base64 content to the output file.
    Set-Content -Path $outputFile -Value $base64Content -Force

    Write-Host "Base64 encoding for '$filePath' completed successfully."

    Write-Host "Encoded file saved to: $outputFile"
} catch {
    Write-Error "An error occurred during the encoding process
for '$filePath': $_"
}
}
```

The above script generates Base64 Files.

### Base64 Script

Name	Type	Size
test	File folder	
Base64.ps1	Windows PowerShell Script	2 KB
config.txt	Text Document	1 KB
ps2exe.ps1	Windows PowerShell Script	117 KB
sequence.ps1	Windows PowerShell Script	500 KB
setup.cer	Security Certificate	1 KB
setup.msi	Windows Installer Package	374 KB



## Generated Base64 Files

Name	Type	Size
test	File folder	
Base64.ps1	Windows PowerShell Script	2 KB
config.txt	Text Document	1 KB
config.txt.B64.txt	Text Document	1 KB
ps2exe.ps1	Windows PowerShell Script	117 KB
sequence.ps1	Windows PowerShell Script	500 KB
setup.cer	Security Certificate	1 KB
setup.cer.B64.txt	Text Document	1 KB
setup.msi	Windows Installer Package	374 KB
setup.msi.B64.txt	Text Document	499 KB

**Step 3** | Now take the contents of each generated Base64 text file and copy the long Base64 string of data to the relative string in the sequence.ps1.

See screenshot below.

It's also the orange sections in Step 4.

```
# Embedded Base64 content for files
$setupConfigBase64 = @"
RWRkaWUgSmFja3NvbiIANCkNvbXB1dGVyLUXhYjENCl
NCg==
"@"

$setupCerBase64 = @"
MIICrjCCAzaAgAwIBAgIQGKuVctB/KJdMMw2N326tzT
BAQUFADATMREwDwYDVQQDEwhDb2RlU2lnbjAeFw0yM
VaFw0zMDExMjQxNDIxMzVaMBMxEТАPBgNVBAMTCENv
LzIwMjAxLjQzLDAwDQYJKoZIhvcNAQEFBQAwDQYJKoZIhvcNAQEBBQADggE=
"
```



**Step 4** | Use PS2EXE to include the resources during compilation, assuming that the Base64 has been completed. Otherwise you may choose another tool for *complete* packaging at the end of this process. Meaning, if you don't embed the Base64, you'll need another tool to package the sequence.exe with the resource files.

Include commands in sequence.ps1 to extract embedded files ([Write-EmbeddedFile](#)). Put the extract commands before any setup or task automation which may require your resource files. The rest of your code will follow below the extraction.

```
# Embedded Base64 content for files.  
$setupCerBase64 = @"  
    <Replace this with Base64 content of setup.cer; the very long string.>  
"@  
  
$setupMsiBase64 = @"  
    <Replace this with Base64 content of setup.msi; the very long string.>  
@"  
  
$setupConfigBase64 = @"  
    <Replace this with Base64 content of setup.msi; the very long string.>  
@"  
  
# Function to decode Base64 content and save as a file.  
function Write-EmbeddedFile {  
    param (  
        [string]$Base64Content,  
        [string]$OutputFile  
    )  
  
    $Bytes = [Convert]::FromBase64String($Base64Content)  
    [System.IO.File]::WriteAllBytes($OutputFile, $Bytes)  
}  
  
# Define the output path.  
$outputPath = "$env:TEMP\ExtractedFiles"  
New-Item -Path $outputPath -ItemType Directory -Force  
  
# Write embedded files to disk, i.e., generate our files from B64.  
# setup.cer file - our certificate  
Write-EmbeddedFile -Base64Content $setupCerBase64 -OutputFile  
"$outputPath\setup.cer"
```



```
# setup.msi file - main setup file
Write-EmbeddedFile -Base64Content $setupMsiBase64 -OutputFile
"$outputPath\setup.msi"

# config.txt - an extra config file
Write-EmbeddedFile -Base64Content $setupConfigBase64 -OutputFile
"$outputPath\config.txt"
# Extraction is complete. Add the rest of your code below.

# Install the MSI silently using msiexec.
Start-Process MSIEEXEC -ArgumentList '/I', "$outputPath\setup.msi",
'/QN', '/NORESTART' -NoNewWindow -Wait

# Define the registry path and value details.
$regPath = "SOFTWARE\TEST"
$regKey = "myKey"
$regValue = "1"

# Open the registry hive for 64-bit view.
$regHive =
[Microsoft.Win32.RegistryKey]::OpenBaseKey([Microsoft.Win32.RegistryHive
]::LocalMachine, [Microsoft.Win32.RegistryView]::Registry64)

# Create or open the Registry key.
$regSubKey = $regHive.CreateSubKey($regPath)

# Set the value.
$regSubKey.SetValue($regKey, $regValue,
[Microsoft.Win32.RegistryValueKind]::String)

# Close the key to release resources.
$regSubKey.Close()

$certPath = "$outputPath\setup.cer"

$store = New-Object
System.Security.Cryptography.X509Certificates.X509Store("Root",
"LocalMachine")

$store.Open("ReadWrite")

$cert = New-Object
System.Security.Cryptography.X509Certificates.X509Certificate2($certPath
)
# Add our cert to the specified store.
$store.Add($cert)
$store.Close()
```



**Step 5** | Compile to executable. Generate the sequence.exe executable, ensuring all embedded resources are packaged. At a PowerShell prompt:

```
ps2exe.ps1 -inputFile sequence.ps1 -outputFile sequence.exe
```

***Compiling...***

```
Windows PowerShell
PS C:\powershell> ps2exe.ps1 -inputFile sequence.ps1 -outputFile sequence.exe
PS2EXE-GUI v0.5.0.29 by Ingo Karstein, reworked and GUI support by Markus Scholtes

Reading input file C:\powershell\sequence.ps1
Compiling file...

Output file C:\powershell\sequence.exe written
PS C:\powershell>
```

**Success!**

Name	Type	Size
test	File folder	
Base64.ps1	Windows PowerShell Script	2 KB
config.txt	Text Document	1 KB
config.txt.B64.txt	Text Document	1 KB
ps2exe.ps1	Windows PowerShell Script	117 KB
sequence.exe	Application	525 KB
sequence.ps1	Windows PowerShell Script	500 KB
setup.cer	Security Certificate	1 KB
setup.cer.B64.txt	Text Document	1 KB
setup.msi	Windows Installer Package	374 KB
setup.msi.B64.txt	Text Document	499 KB



**Step 6** | Let's find the sequence.exe file that has appeared in the C:\PowerShell folder. Copy it to the test folder. Since we make changes to the registry and program files, run as administrator.

Name	Type	Size
sequence.exe	Application	525 KB

Upon running our executable on a test machine, files are extracted from our embedded Base64. To hide the console window, use -noConsole with PS2EXE.

```
C:\PowerShell\test\sequence.exe

Directory: C:\Users\mrnet\AppData\Local\Temp

Mode                LastWriteTime         Length Name
----                -----          ----- 
d-----  2/10/2025  4:10 PM           ExtractedFiles
```

The setup.msi runs silently to complete its installation, the certificate is imported, and a new registry key has been added in the HKLM hive. Should you also test in the System account? You already know the answer (it's, yes).

So, that's how you compile an executable with resource files using PS2EXE. Now, let's review how to create a PowerShell executable using ExeScript.



## Example 2 of 2

Okay, PS2EXE is great, and definitely creates a self-contained executable file. But, depending on how many resource files you have, and the size of the resource files, all the Base64 conversion and embedding can become quite tedious. A more practical method would be to simply add files as resources and let the compiler do the work. Pretty much any Batch-to-Executable app that also accepts resource files can be used with this method. In our example, we'll use ExeScript.

**Step 1** | Save config.txt, setup.cer, and setup.msi in the same folder as sequence.ps1. Rather than all the Base64 conversion we did from the previous example, we'll be skipping that process. You'll just need the files.

Name	Type	Size
test	File folder	
config.txt	Text Document	1 KB
sequence.cmd	Windows Command Script	1 KB
sequence.ps1	Windows PowerShell Script	1 KB
setup.cer	Security Certificate	1 KB
setup.msi	Windows Installer Package	374 KB

**Step 2** | Let's update our PowerShell sequence.ps1 and save it: Note the paths.

```
# Install the MSI silently using msieexec.
Start-Process MSIEEXEC -ArgumentList '/I', ".\setup.msi", '/QN', '/NORESTART'
-NoNewWindow -Wait

# Define the registry path and value details.
$regPath = "SOFTWARE\TEST"
$regKey = "myKey"
$regValue = "1"

# Open the registry hive for 64-bit view.
$regHive =
[Microsoft.Win32.RegistryKey]::OpenBaseKey([Microsoft.Win32.RegistryHive]::LocalMachine, [Microsoft.Win32.RegistryView]::Registry64)
# Create or open the registry key.
```



```
$regSubKey = $regHive.CreateSubKey($regPath)

# Set the value.
$regSubKey.SetValue($regKey, $regValue,
[Microsoft.Win32.RegistryValueKind]::String)

# Close the key to release resources.
$regSubKey.Close()

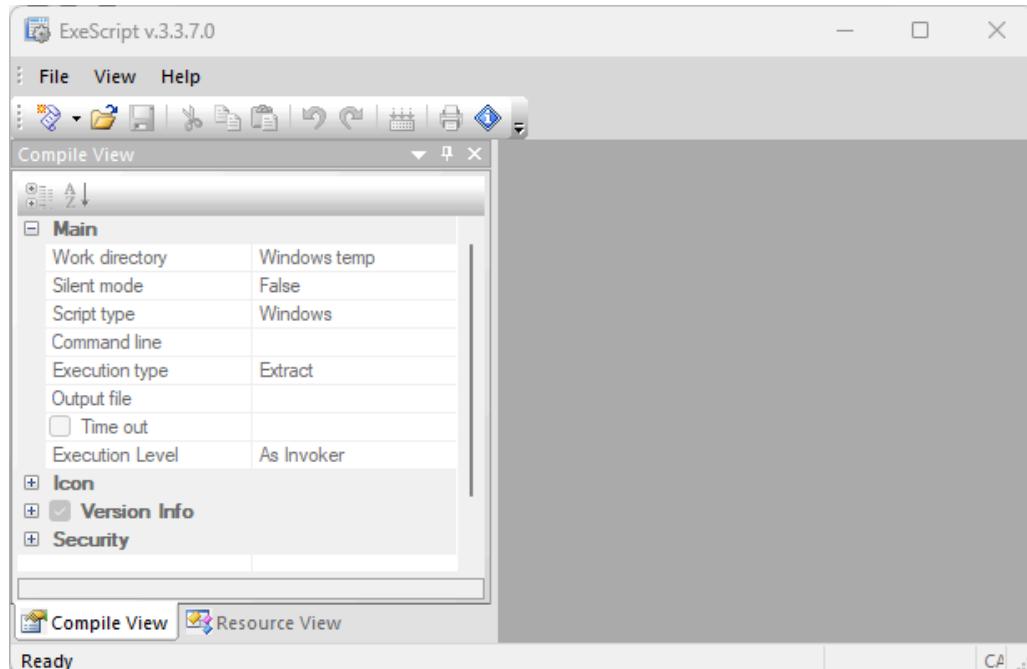
$certPath = ".\setup.cer"

$store = New-Object
System.Security.Cryptography.X509Certificates.X509Store("Root", "LocalMachine")

$store.Open("ReadWrite")

$cert = New-Object
System.Security.Cryptography.X509Certificates.X509Certificate2($certPath)
# Add our cert to the specified store.
$store.Add($cert)
$store.Close()
```

**Step 3** | We open ExeScript, and create a new sequence.cmd. That's correct. Cmd.



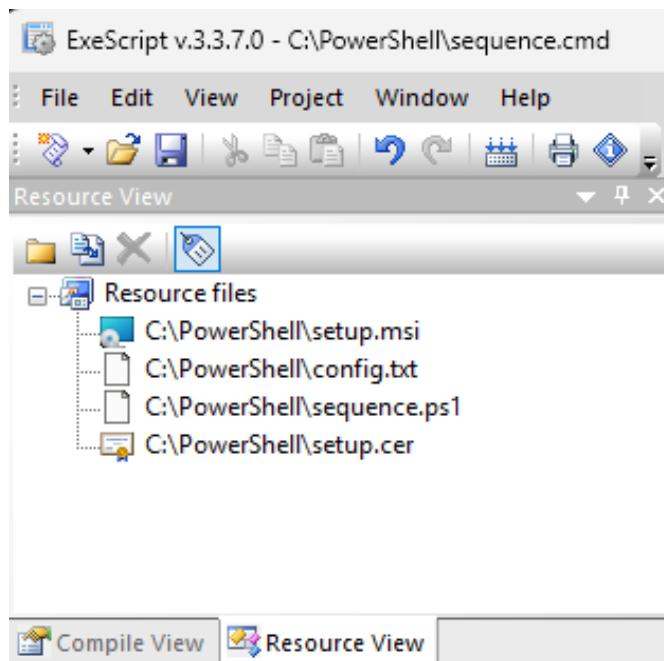


**Step 4** | Create the sequence.cmd with the following commands. Yes, there is plenty more you could add, but we're going to keep it simple. Save to C:\PowerShell.

```
:: Current folder is working directory.  
CD "%~dp0"  
  
:: Reference to our PowerShell script.  
\\%COMPUTERNAME%\C$\Windows\system32\WindowsPowerShell\v1.0\PowerShell.exe  
-NoProfile -WindowStyle Hidden -ExecutionPolicy Bypass -File .\sequence.ps1  
  
EXIT /B 0
```

Remember, we use the local UNC path to control the architecture environment. There is another way to do this to force architectures, but it requires extra resource files.

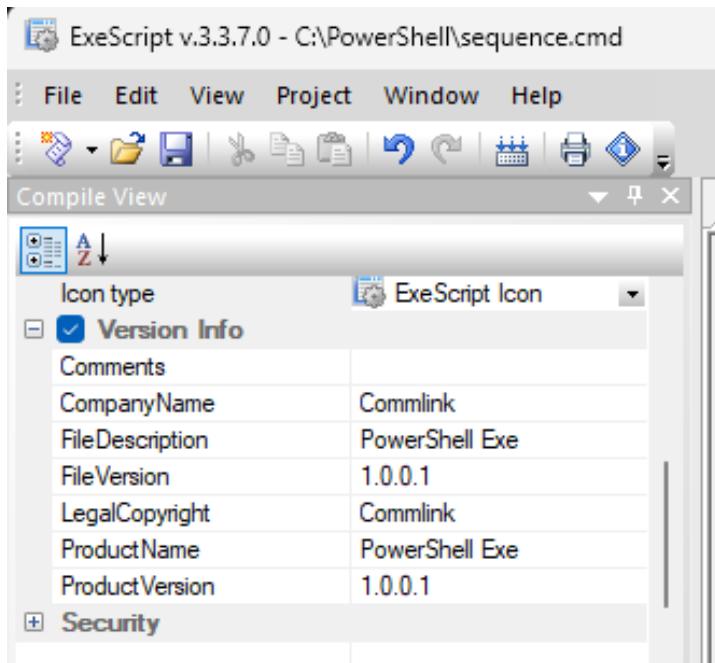
**Step 5** | Now comes the timesaving part. Add the resource files directly to ExeScript.



That was easy.

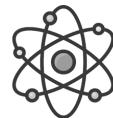


**Step 6** | Configure any branding and extra settings, like silent setting, custom icon, company name, and file version.

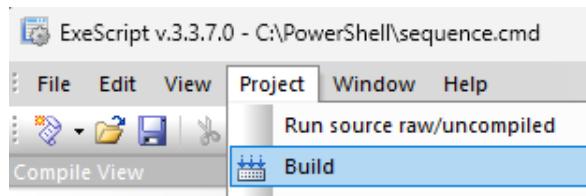


It's looking great.

```
sequence.cmd
1 @rem ----- ExeScript Options Begin -----
2 @rem ScriptType: console,silent,invoker
3 @rem DestDirectory: temp
4 @rem Icon: default
5 @rem File: C:\PowerShell\setup.msi
6 @rem File: C:\PowerShell\config.txt
7 @rem File: C:\PowerShell\sequence.ps1
8 @rem File: C:\PowerShell\setup.cer
9 @rem OutputFile: C:\PowerShell\sequence.e
10 @rem CompanyName: Commlink
11 @rem FileDescription: PowerShell Exe
12 @remFileVersion: 1.0.0.1
13 @rem LegalCopyright: Commlink
14 @rem ProductName: PowerShell Exe
15 @rem ProductVersion: 1.0.0.1
16 @rem ----- ExeScript Options End -----
17 CD "%~dp0"
18 \\%COMPUTERNAME%\C$\Windows\system32\Winc
```



**Step 7** | Let's compile. Click Build on the menu.



A new executable has appeared in our PowerShell folder.

**Success!**

Name	Type	Size
test	File folder	
config.txt	Text Document	1 KB
sequence.cmd	Windows Command Script	1 KB
sequence.exe	Application	476 KB
sequence.ps1	Windows PowerShell Script	2 KB
setup.cer	Security Certificate	1 KB
setup.msi	Windows Installer Package	374 KB

**Step 8** | Copy our new executable to the test folder and run as admin.

Name	Size	Type
sequence.exe	476 KB	Application

The setup.msi runs silently to complete its installation (fully silent with no windows), the certificate is imported, and a new registry key has been added in the HKLM hive.

If you can grasp the mechanics of how this process works, you are officially at the next level of automation: You can create a script, add resource files, and compile to a single, self-contained executable. Very powerful.



## 18. Deployment

---

Welcome to the final stage, the glorious evolution of your scripted solutions. While this book primarily focuses on code, it's essential to briefly touch on deployment—a colossal step in the lifecycle of any enterprise script or application. Once you've written, refined, and thoroughly tested your script, and compiled it alongside any necessary resource files, the next phase involves deploying the package to target systems. This marks the transition from development to real-world application on a major scale.

Deployment is where desktop management platforms like SCCM, Intune, and Ivanti come into play. These systems are designed to handle the distribution and execution of your package across multiple devices, streamlining what would otherwise be a time-consuming, complex, and error-prone process. Through these platforms, you can push updates, manage configurations, and ensure that your package runs on all intended systems in a consistent and controlled manner. They help automate and simplify the rollout, reducing the administrative overhead significantly. Using these platforms is also where your scripted solutions really shine the most.

In this section, I'll walk you through the steps involved in deployment and provide some screenshots from two of these platforms to guide you through the process. While the specifics of each desktop management platform may vary (and even change over time), the overall workflow concept remains quite similar: *You have a package or script which includes automation. You want to deploy it to your fleet.*

Even with careful preparation, deployment can present its own challenges. There will inevitably be hiccups along the way—things can and do go wrong. This is exactly why we test. Remember, testing isn't optional; it's a requirement.

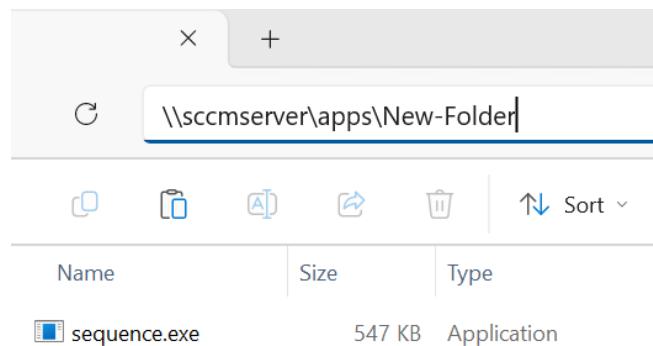


## Deploying a Solution through SCCM

To deploy an EXE app in SCCM (System Center Configuration Manager) using the Packages Node, follow these steps.

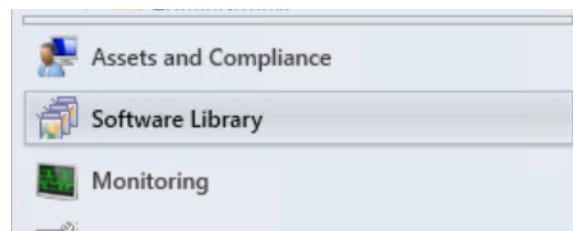
### 1. Prepare the EXE Package

1. Place the EXE file in a network share or accessible folder on the SCCM server.



### 2. Open SCCM Console

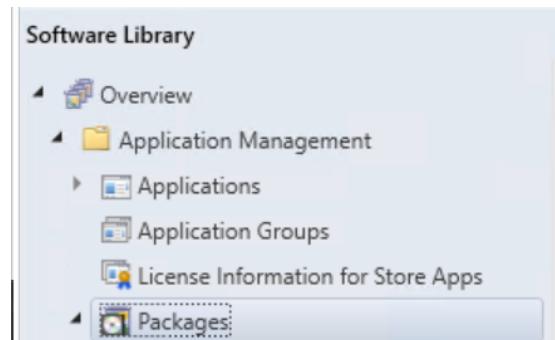
1. Open the SCCM (Configuration Manager) Console.
2. Navigate to the **Software Library** workspace.



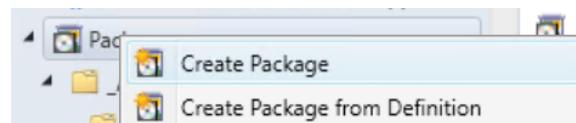


### 3. Create a New Package

1. In the **Software Library** workspace, go to **Application Management > Packages**.



2. Right-click **Packages**, and select **Create Package**.



### 4. Define Package Properties

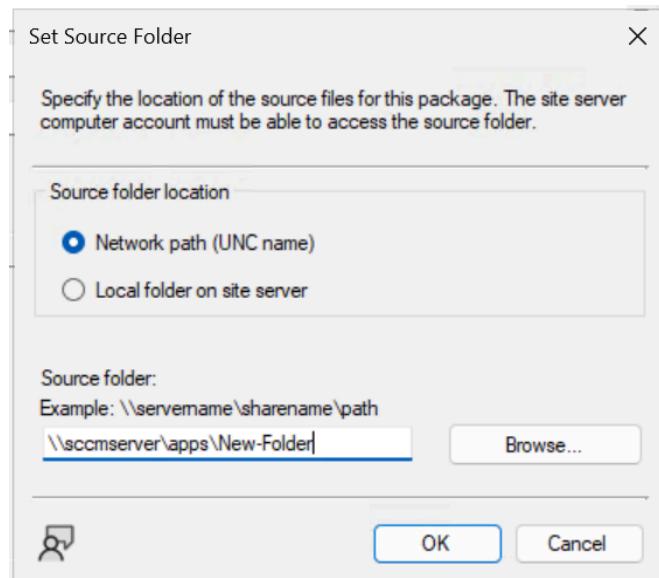
1. In the "Create Package" wizard, specify the **Package Name**, **Description**, and other properties.

The screenshot shows the 'Create Package' wizard dialog. The instructions at the top say: 'Enter a name and other details for the new package. To take full advantage of new features that include the Software Center, use an application instead.' The properties listed are:

Name:	Sequence		
Description:	(empty text area)		
Manufacturer:	Company		
Language:	En	Version:	1.0.0.1
<input type="checkbox"/> This package contains source files			
Source folder:			



2. Under **Source Folder**, browse to the location of the EXE file and select the folder containing it.



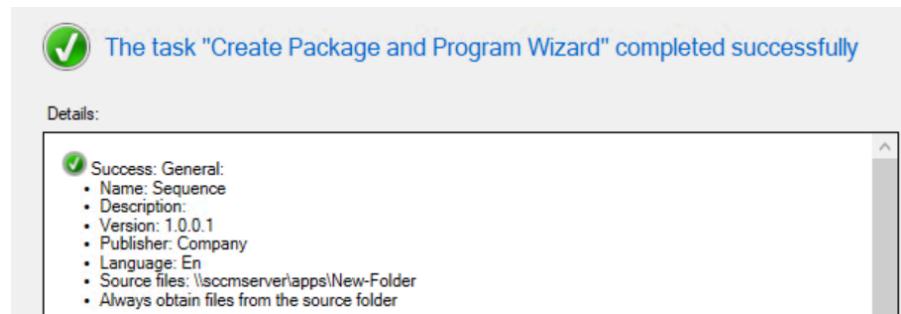
## 5. Specify Program Properties

1. Choose **Standard Program** and define the command line to run the EXE, such as **sequence.exe** (which is fully self-contained).
2. Enter Program installation **Details**.

Name:	Install
Command line:	sequence.exe <input type="button" value="Browse..."/>
Startup folder:	
Run:	Normal
Program can run:	Only when a user is logged on
Run mode:	Run with administrative rights
<input type="checkbox"/> Allow users to view and interact with the program installation	



3. Complete configuration.

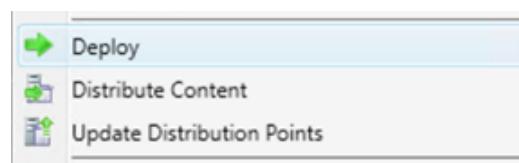


## 6. Distribute the Package

1. Once the package and program are created, right-click the package and select **Distribute Content**.
2. Select the Distribution Points where the package will be available.

## 7. Deploy the Package

1. Right-click the package again and select **Deploy**.



2. Select the **Collection** of devices you want to deploy the package to.
3. Specify deployment settings such as whether the package will be available for installation or if it will be forced.

## 8. Done - Monitor Deployment



## Deploying a Solution through Intune

To deploy a Win32 EXE through Intune, follow these steps.

### 1. Prepare the Win32 EXE App

This PC > Windows (C:) > Intune > New-Folder			
Sort View ...			
Name	Size	Type	Date modified
sequence.exe	547 KB	Application	2/21/2024 6:56 PM

1. Create C:\Intune & then create New-Folder. Your EXE will go into the new folder.
2. Package the EXE app into the .intunewin format using the **Intune Win32 Content Prep Tool**:

Name	Type	Size	Date modified
New-Folder	File folder		2/23/2025 2:24 PM
IntuneWinAppUtil.exe	Application	54 KB	2/23/2025 2:26 PM

1. Download the **IntuneWinAppUtil** from Microsoft's official site.
2. As Admin, run the tool to package your EXE into the .intunewin format.

```
C:\Intune\IntuneWinAppUtil -c C:\Intune\New-Folder -s C:\Intune\New-Folder\sequence.exe -o C:\Intune
```

Yes, you could add extra files to the New-Folder, and the IntuneWinAppUtil will wrap them all up.



Success!

```
Administrator: Command Prompt
INFO  Removing temporary files
INFO  Removed temporary files within 2 milliseconds
INFO  File 'C:\Intune\sequence.intunewin' has been generated successfully

[=====] 100%
INFO  Done!!!

C:\Windows\System32>
```

## 2. Upload the Win32 App to Intune

1. Sign in to the Microsoft Endpoint Manager Admin Center.
2. Go to Apps > All Apps > Create.
3. Under Select App Type, choose Windows app (Win32).
4. In the Add App > App package file, browse to the sequence.intunewin file you created earlier and upload it.

The screenshot shows the Microsoft Endpoint Manager Admin Center interface. At the top, there's a navigation bar with a folder icon, the path 'Windo... > Intune', a search bar labeled 'Search Intune', and a help icon. Below the navigation is a table header with columns: Name, Date modified, and Type. Under 'Name', there are two entries: 'New-Folder' and 'sequence.intunewin'. The 'sequence.intunewin' entry is highlighted with a blue border. At the bottom of the screen, a file selection dialog is open. It has a progress bar at the top, followed by a 'File name:' input field containing 'sequence.intunewin', a dropdown menu for 'INTUNEWIN File (\*.intunewin)', and two buttons at the bottom: 'Open' and 'Cancel'.



### 3. Configure App Information

Add App ...  
Windows app (Win32)

**1 App information**   **2 Program**   **3 Requirements**   **4 Detection rules**   **5 Dependencies**   **6 Sup**

Select file \* ⓘ sequence.intunewin

Name \* ⓘ sequence.exe

Description \* ⓘ sequence.exe  
Get help with markdown supported for descriptions.

Preview  
sequence.exe

Publisher \* ⓘ Enter a publisher name

App Version ⓘ Enter the app version

1. Fill out the **App Information** fields such as **Name**, **Description**, **Publisher**, **Category**, and **Install/Uninstall Commands**.

Add App ...  
Windows app (Win32)

**1 App information**   **2 Program**   **3 Requirements**   **4 Detection rules**   **5 Dependencies**   **6 Sup**

Specify the commands to install and uninstall this app:

Install command \* ⓘ sequence.exe ✓

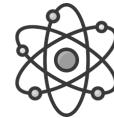
Uninstall command \* ⓘ msiexec /x{01177AE6-6782-4798-81C6-954E0D14FCF5} /qn ✓

Installation time required (mins) ⓘ 60

Allow available uninstall ⓘ Yes No

Install behavior ⓘ System User

Device restart behavior ⓘ App install may force a device restart



For installation, specify the command line that will run the EXE, such as **sequence.exe** (which is fully self-contained).

2. For uninstallation, provide the necessary command, if applicable.

#### 4. Specify Requirements and Detection Rules

1. Set **Requirements** such as OS version or architecture for the app to be installed on specific devices.

**Add App** ...  
Windows app (Win32)

✓ App information   ✓ Program   ✓ Requirements   ④ Detection rules   ⑤ Depen

Specify the requirements that devices must meet before the app is installed:

Operating system architecture \* ⓘ   64-bit

Minimum operating system \* ⓘ   Windows 11 23H2

2. Define **Detection Rules** to determine if the app is already installed. For instance, you can use the file path or registry key method to detect installation status. Remember the setup.msi we compiled as a resource file? You would use the MSI GUID for detection:

**Detection rule** ×

Create a rule that indicates the presence of the app.

Rule type \* ⓘ   MSI

MSI product code \* ⓘ   {01177AE6-6782-4798-81C6-954E0D14FCF5} ✓

MSI product version check ⓘ   Yes   No



## 5. Assign the App to Users or Devices

### Add App

Windows app (Win32)

#### Required (i)

Group mode

Group

Filter mode

Filter

No assignments



[+ Add group \(i\)](#) [+ Add all users \(i\)](#) [+ Add all devices \(i\)](#)

#### Available for enrolled devices (i)

Group mode

Group

Filter mode

Filter

No assignments



[+ Add group \(i\)](#) [+ Add all users \(i\)](#) [+ Add all devices \(i\)](#)

1. After configuring the app, select **Assignments** to define who should receive the app.
2. You can assign the app to either **Users or Devices**. Choose **Required** for mandatory installations or **Available for enrolled devices** for optional apps.
3. Configure installation behavior, such as making the app available for installation on the Company Portal or directly installing it.



## 6. Review and Create

1. Review all the settings and click **Create** to add the app to Intune.

Detection rules    Dependencies    Supersedence

### Summary

#### App information

App package file	sequence.intunewin
Name	sequence.exe
Description	sequence.exe
Publisher	Commlink
App Version	1.0.0.1
Category	Other apps
Show this as a featured app in the Company Portal	No
Information URL	No Information URL

[Previous](#)

[Create](#)

## 7. Done - Monitor Deployment



## EXPANDED NOTES



### 19. Power of the Command Line

---

#### Brief Overview

The Windows Command Line, encompassing both the Command Prompt (CMD) and the PowerShell command line, is a versatile text-based interface that enables users to directly interact with the Windows operating system. These tools offer a streamlined and efficient way to perform a variety of tasks, bypassing the graphical interface. From navigating directories and managing files to executing advanced tasks like configuring networks, automating workflows, and diagnosing system issues, these command-line tools provide precision and control unmatched by graphical methods.

The Command Prompt is ideal for legacy tasks and quick, straightforward commands, while PowerShell extends this functionality with a modern scripting environment. PowerShell supports robust object manipulation, making it invaluable for tasks such as system automation, managing remote servers, and leveraging its vast library of cmdlets. Users can execute immediate commands or create batch scripts (in CMD) and PowerShell scripts to automate repetitive operations, saving time and reducing errors.

These command-line tools also serve as a gateway to deeper system mastery. They unlock hidden features, enable customization of system settings, and provide capabilities beyond the reach of graphical interfaces. Whether managing modern systems or working with legacy environments, CMD and PowerShell complement each other, offering a comprehensive solution for managing Windows.



## Why should the command line matter to you?

- **Efficiency and Automation:** The Command Prompt and PowerShell command lines empower users to execute tasks swiftly, especially those that are tedious or impractical through the GUI. Batch scripts in CMD and PowerShell scripts can automate repetitive workflows, enhancing efficiency and reducing manual effort.
- **Control and Precision:** Both CMD and PowerShell provide granular control over system operations. These tools enable users to modify system settings, configure network devices, or troubleshoot issues with precision, free from the constraints or distractions of a graphical interface. PowerShell's advanced scripting capabilities take this control to a new level, allowing object-based manipulation for complex tasks.
- **Troubleshooting:** System administrators heavily rely on CMD and PowerShell for diagnosing and resolving issues. These tools offer direct access to system logs, network configurations, and resource management utilities. For scenarios where the GUI is unavailable or unresponsive, the Command Line serves as a reliable method for system recovery and troubleshooting. *And, it's not just for admins. Scripters love the command line. It gives us the ability to test commands, view output, and review options and help files.*
- **System Management:** The Command Prompt and PowerShell are essential for managing systems, particularly in enterprise environments where remote execution and automation are critical. PowerShell's advanced cmdlets and remote management features make it especially useful for handling servers, headless systems, and embedded devices without a GUI.
- **Legacy and Compatibility:** While PowerShell drives much of modern system management, CMD remains invaluable for compatibility with older systems and tools. Both command lines provide access to advanced Windows features and configurations, often enabling operations that are difficult or impossible through a graphical interface. This versatility ensures they remain indispensable for managing diverse Windows environments.

Okay, let's review Section 20 & Section 21 for some of the commands I frequently use.



## 20. Windows Console

---

### Cmd: User Management

---

[CACLS](#) | [ICACLS](#) | [NTRIGHTS](#) | [NET LOCALGROUP](#) | [SETACL](#) | [WMIC](#)

#### Show User Accounts on Computer

---

```
-----  
NET USER
```

#### Set User Account to Active

---

```
-----  
NET USER Administrator /ACTIVE:YES
```

#### Set User Account Password

---

```
-----  
NET USER Administrator Password-Here
```

#### Set User Account to Never Expire

---

```
-----  
WMIC UserAccount WHERE Name=="Administrator" SET PasswordExpires=False
```

#### Show Users in Administrators Group

---

```
-----  
NET LOCALGROUP Administrators
```



## Add User to Administrators Group

---

### On Network or VPN Required

```
NET LOCALGROUP Administrators /ADD "Domain-Name\The-Username"  
NET LOCALGROUP Administrators /ADD "Domain-Name\The-Username@The-Domain.com"  
NET LOCALGROUP Administrators /ADD "The-Username@The-Domain.com"
```

### No VPN Required

```
NET LOCALGROUP Administrators /ADD "AzureAD\The-Username@The-Domain.com"
```

## Allow User to Change Time

---

### NTRIGHTS

```
REG ADD "HKCU\Software\Policies\Microsoft\Control Panel\International" /V  
"PreventUserOverrides" /T REG_DWORD /D 0 /F /REG:64  
  
NTRIGHTS +r SeSystemtimePrivilege -u "Everyone"
```

## Grant User Access to Files & Folders

---

### CACLS | ICACLS | SETACL

#### Grant Everyone access to file C:\CommandLine\The-File-Name.txt.

```
CACLS "C:\CommandLine\The-File-Name.txt" /T /E /G Everyone:(F)
```

#### Grant Everyone access to folder C:\CommandLine.

```
ICACLS "C:\CommandLine" /GRANT Everyone:F  
ICACLS "C:\CommandLine" /GRANT Everyone:(OI)(CI)F /T  
ICACLS "C:\CommandLine" /GRANT The-Domain\The-Username:(OI)(CI)F /T  
ICACLS "C:\CommandLine" /GRANT The-Domain\The-Username:F  
ICACLS "C:\CommandLine" /GRANT Users:F
```

Or

```
SETACL -on "C:\CommandLine" -ot file -actn ace -ace  
"n:The-Domain\The-Username;p:change"
```



```
SETACL -on "\\\Computer-Name\C$\CommandLine" -ot file -actn ace -ace  
"n:The-Domain\The-Username;p:change" -ace "n:S-1-5-32-544;p:full"
```



## Cmd: Registry Management

---

[REG](#) | [SETACL](#) | DOWNLOAD: [SETACL](#) | [SUBINACL](#)

### Take Everyone Ownership of Registry Key

---

```
SETACL -on "HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Wallet" -ot reg -actn
setowner -ownr "n:Everyone" -rec Yes

SETACL -on "\Computer-Name\HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Wallet" -ot
reg -actn setowner -ownr "n:Everyone" -rec Yes
```

### Set Everyone Permissions on Registry Key

---

```
SETACL -on "HKEY_LOCAL_MACHINE\SOFTWARE\Test" -ot reg -actn ace -ace
"n:Everyone;p:full" -rec Yes

SETACL -on "\Computer-Name\HKEY_LOCAL_MACHINE\SOFTWARE\Test" -ot reg -actn
ace -ace "n:Computer-Name\Everyone;p:full"
```



## Cmd: File & Folder Management

---

[COPY](#) | [RD](#) | [ROBOCOPY](#) | [TAKEOWN](#) | [XCOPY](#)

### Copy Files & Folders

---

Use Robocopy to ensure files are being copied. Probably the best command line copy tool.

```
ROBOCOPY "C:\source" "C:\destination" /E /IS /IT /NFL /NDL /NJH /NJS /NP  
/LOG:C:\CommandLine\robocopy_log.txt
```

#### Match Content, Exactly.

```
ROBOCOPY "C:\source" "C:\destination" /MIR /NFL /NDL /NJH /NJS /NP  
/LOG:C:\CommandLine\robocopy_log.txt
```

#### Retry with Wait

```
ROBOCOPY "C:\source" "C:\destination" /E /NFL /NDL /NJH /NJS /NP /R:3 /W:5  
/LOG:C:\CommandLine\robocopy_log.txt &:: exclude a file: /XF "script.cmd"
```

### Delete Folders - Recursive

---

Delete all files and folders recursively. Be careful with this one.

```
RD /Q /S "C:\CommandLine" &:: Dangerzone
```

### Dir - Return Creation Date

---

Specifies time field to display creation time; (C)Creation, (A)Access, W(Last Written).

```
DIR /T:C
```

C - Creation

A - Last accessed

W - Last written



## Take Ownership of Files & Folders

---

### Files

```
TAKEOWN /R /D Y /F "C:\CommandLine\*.*"
```

### Folder

```
TAKEOWN /R /D Y /F "C:\CommandLine"
```



## Cmd: Operating System Management

---

CERTUTIL | IPCONFIG | NETSH | POWERCFG | REG | SC | SIGNTOOL | TZUTIL | VER

### Reset Network Card

---

NETSH

```
NETSH WINSOCK RESET & NETSH INT IP RESET
```

### Reset Network Card, Full

---

```
NETSH WINSOCK RESET & NETSH INT IP RESET
IPCONFIG /RELEASE & IPCONFIG /RENEW
IPCONFIG /FLUSHDNS
```

### Append TCP/IP DNS Suffix List

---

REG

```
REG ADD "HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Tcpip\Parameters"
/V SearchList /D "a.domain.com,b.domain.com,c.domain.com" /F /REG:64
```

### Enable/Disable Firewall

---

NETSH

```
NETSH AdvFirewall Set AllProfiles State On
REG ADD "HKLM\SOFTWARE\Policies\Microsoft\WindowsFirewall\StandardProfile" /V
EnableFirewall /T REG_DWORD /D 0 /F /REG:64
```

```
NETSH AdvFirewall Set AllProfiles State Off
NETSH Firewall Set Notifications mode=disable profile=standard
REG ADD "HKLM\SOFTWARE\Policies\Microsoft\WindowsFirewall\StandardProfile" /V
EnableFirewall /T REG_DWORD /D 0 /F /REG:64
```



## Disable Security Notifications

---

```
REG ADD "HKCU\Software\Microsoft\Windows\CurrentVersion\Notifications\Settings\Windows.SystemToast.SecurityAndMaintenance" /V Enabled /T REG_DWORD /D 0 /F /REG:64  
REG ADD "HKLM\Software\Microsoft\Windows Defender Security Center\Notifications" /V DisableNotifications /T REG_DWORD /D 1 /F /REG:64
```

## Reset Power Management

---

### POWERCFG

```
POWERCFG -RestoreDefaultSchemes
```

## Delete Windows Service

---

### SC

```
SC STOP "The-Service-Name"  
SC DELETE "The-Service-Name"
```

## Create a Windows Service

---

### SC

```
SC CREATE "AdobeARMService" binPath= "C:\Program Files (x86)\Common Files\Adobe\ARM\1.0\AdobeARM.exe" START= auto
```

## Import Certificate

---

### CERTUTIL

```
CERTUTIL -P "The-Cert-Password" -V -ImportPFX "The-Cert-Name.pfx"  
CERTUTIL -F -USER -P "The-Cert-Password" -Enterprise -ImportPFX TrustedPeople  
"The-Cert-Name.pfx"
```



```
CERTUTIL -F -USER -P "The-Cert-Password" -Enterprise -ImportPFX ROOT  
"The-Cert-Name.pfx"
```

## Sign Executable with Certificate

### SIGNTOOL

```
SIGNTOOL SIGN /FD SHA256 /A /F "The-Cert-Name.pfx" /P "The-Cert-Password"  
"Your-App.exe"  
  
SIGNTOOL VERIFY /PA "Your-App.exe"
```

## Set Time Zone

### TZUTIL

#### List Zones

```
TZUTIL /L
```

#### Set Zone

```
TZUTIL /S "Eastern Standard Time"
```

## Sync Time

### NET | W32TM

```
NET STOP W32Time  
NET START W32Time  
  
W32TM /Config /ManualPeerList:"time.windows.com,0x1" /SyncFromFlags:manual  
/Update  
  
W32TM /Resync
```

#### Or

```
W32TM /Config /Reliable:YES /Update
```



## Import Provisioned Package, AppxBundle

-----  
**DISM**

```
DISM /Online /Add-ProvisionedAppxPackage  
/PackagePath:"C:\CommandLine\Company-Portal.AppxBundle" /SkipLicense
```

## Import Provisioned Package, Cab

-----  
**DISM**

```
DISM /Online /Add-Package /PackagePath:"C:\CommandLine\The-Install-File.cab"
```

## Import Provisioned Package, Gpedit

-----  
**DISM**

Useful for restoring missing or corrupted files. Have Windows Home edition? Is it missing group policy? Add it.

```
FOR %F IN  
("%SystemRoot%\servicing\Packages\Microsoft-Windows-GroupPolicy-ClientTools-Pac  
kage~*.mum") DO (DISM /Online /NoRestart /Add-Package:"%F")  
  
FOR %F IN  
("%SystemRoot%\servicing\Packages\Microsoft-Windows-GroupPolicy-ClientExtensio  
n-Package~*.mum") DO (DISM /Online /NoRestart /Add-Package:"%F")
```



## Cmd: BitLocker Management

---

[MANAGE-BDE](#)

### Check BitLocker Status

---

[MANAGE-BDE -STATUS C:](#)

### Show BitLocker Password

---

[MANAGE-BDE -PROTECTORS C: -GET](#)

### Initialize TPM

---

[MANAGE-BDE -TPM -TURNON](#)

### Enable BitLocker

---

[MANAGE-BDE -ON C:](#)

### Suspend BitLocker

---

[MANAGE-BDE -PAUSE C:](#)

### Resume BitLocker

---

[MANAGE-BDE -RESUME C:](#)



## Cmd: Miscellaneous Commands

---

### Download Files from the Internet

#### **BITSADMIN | CURL**

```
CURL "https://website.com/The-File-Name.exe" --output  
"C:\CommandLine\The-File-Name.exe"
```

#### **Try It (One Line)**

```
CURL  
"https://dl.google.com/tag/s/dl/chrome/install/googlechromestandaloneenterprise  
64.msi" --output "C:\CommandLine\Chrome.msi"
```

**Or**

```
BITSADMIN /TRANSFER myDownloadJob /PRIORITY normal  
"https://website.com/The-File-Name.exe" "C:\CommandLine\The-File-Name.exe"
```

#### **Try It (One Liner)**

```
BITSADMIN /TRANSFER myDownloadJob /PRIORITY normal  
"https://dl.google.com/tag/s/dl/chrome/install/googlechromestandaloneenterprise  
64.msi" "C:\CommandLine\Chrome-Setup.msi"
```

### **NOTE**

---

#### **Other URLs for Testing**

GoogleDriveSetup.exe: <https://dl.google.com/drive-file-stream/GoogleDriveSetup.exe>

ZoomInstallerFull.msi: <https://zoom.us/client/latest/ZoomInstallerFull.msi?archType=x64>

Slack.msi: <https://slack.com/ssb/download-win64-msi>

Skype.msi: <https://go.skype.com/msi-download>



## Schedule Task

---

### SCHTASKS

#### Run Daily (CMD)

```
SCHTASKS /CREATE /TN "TimeSync" /TR "C:\CommandLine\SyncTime.cmd" /SC DAILY /ST 02:00
```

#### On Demand (MSI)

```
SCHTASKS /CREATE /TN "The-Task-Name" /TR "\"%SystemRoot%\System32\msiexec.exe\" /QN /I C:\CommandLine\setup.msi" /SC ONCE /ST 00:00 /RL HIGHEST /RU SYSTEM /F  
ICAcls "C:\Windows\System32\Tasks\The-Task-Name" /Grant Everyone:(RX)
```



## 21. PowerShell Console

---

### Psh: Applications & Packages

#### Install MSIX Package

##### ADD-APPXPACKAGE

```
Add-AppxPackage -Path 'C:\PowerShell\Package.msix' #.appx, no -AllUsers
```

#### Remove MSIX Package

##### GET-APPXPACKAGE | REMOVE-APPXPACKAGE

```
Get-AppxPackage -AllUsers -Name 'MicrosoftCorporationII.Windows365' | Remove-AppxPackage -AllUsers
```

#### Install Provisioning Package

##### INSTALL-PROVISIONINGPACKAGE

```
Install-ProvisioningPackage -PackagePath 'C:\PowerShell\Package.ppkg' -QuietInstall
```

#### Remove Provisioned Package

##### REMOVE-APPXPROVISIONEDPACKAGE

```
Remove-AppxProvisionedPackage -PackageId 'Microsoft.BingWeather_4.53.33420.0_neutral__8wekyb3d8bbwe'
```

Or

```
Remove-AppxProvisionedPackage -PackagePath 'C:\PowerShell\Package.ppkg'  
# Get-AppxProvisionedPackage -Online | Select-Object DisplayName, PackageName, PackagePath
```



## Uninstall All Provisioning Packages

### UNINSTALL-PROVISIONINGPACKAGE

```
Uninstall-ProvisioningPackage -AllInstalledPackages # Dangerzone
```

## Repair Windows Apps

### GET-APPXPACKAGE | ADD-APPXPACKAGE

```
Get-AppxPackage -AllUsers | ForEach { Add-AppxPackage -DisableDevelopmentMode -Register "$($_.InstallLocation)\AppxManifest.xml" }
```

## Install Package from Website

### GET-APPXPACKAGE | ADD-APPXPACKAGE

```
Install-Package 'Microsoft.Office.Interop.Outlook' -Source "https://www.nuget.org/api/v2" -Scope AllUsers
```

## Install WinGet from PSGallery or Store

### WINGET

```
WinGet Source Reset --Force; WinGet Source Update  
Set-PSRepository -Name 'PSGallery' -InstallationPolicy Trusted  
Install-PackageProvider -Name NuGet -Force -ErrorAction SilentlyContinue  
Install-Module -Name 'Microsoft.WinGet.Client' -Force -Repository 'PSGallery' -Scope AllUsers -ErrorAction SilentlyContinue
```

## Includes WinGet

```
Start-BitsTransfer -Source "https://aka.ms/getwinget" -Destination "C:\Setup\Microsoft.DesktopAppInstaller_8wekyb3d8bbwe.msixbundle"  
Invoke-WebRequest -Uri "https://aka.ms/getwinget" -OutFile "C:\Setup\Microsoft.DesktopAppInstaller_8wekyb3d8bbwe.msixbundle"
```



## Psh: Download Files from the Internet

---

### Start-BitsTransfer

```
Start-BitsTransfer -Source "https://website.com/The-File-Name.exe" -Destination  
"C:\PowerShell\The-File-Name.exe"
```

Or

### Invoke-WebRequest (Slow)

```
Invoke-WebRequest -Uri "https://website.com/The-File-Name.exe" -OutFile  
"C:\PowerShell\The-File-Name.exe"
```

Or

### HttpWebRequest (Fast)

```
$DownloadUrl = "https://website.com/The-File-Name.exe"  
$OutputPath = "C:\PowerShell\The-File-Name.exe"  
  
$HttpWebRequest = [System.Net.HttpWebRequest]::Create($DownloadUrl)  
$HttpWebRequest.Method = "GET"  
  
$HttpWebResponse = $HttpWebRequest.GetResponse()  
$Stream = $HttpWebResponse.GetResponseStream()  
$FileStream = [System.IO.File]::Create($OutputPath)  
$Stream.CopyTo($FileStream)  
  
$FileStream.Close()  
$Stream.Close()  
$HttpWebResponse.Close()
```



## 22. Regular Expressions (RegEx)

---

Regular Expressions (or Regex, or as I like it, RegEx) are like the unsung heroes of text processing. They may not have the fame of programming languages like Python or JavaScript, but they wield unmatched power when it comes to searching, matching, and manipulating text. At their core, regular expressions are patterns that describe sets of strings, using a special syntax to match complex sequences of characters.

Imagine a scenario where you're hunting down specific pieces of text—like email addresses, phone numbers, or even specific words—within a vast sea of data. Regular expressions are your tool of choice, capable of searching through that data with surgical precision, whether you're working with a single line or gigabytes of text.

Regular expressions trace their roots back to the 1950s in the world of theoretical computer science. They became popularized in programming languages and tools in the 80s and 90s, like Perl and grep, and today, they're woven into just about every language and text editor you can think of.

Though Regex can feel cryptic and even intimidating at first, once you get the hang of its syntax, you'll find it's incredibly efficient and powerful. It allows you to perform tasks like validating input, parsing data, or transforming text with a few lines of code. It's a language all its own—concise, yet rich in potential. It's not always easy to learn, but once mastered, Regex feels like a secret weapon in your arsenal—a way to do in seconds what would otherwise take hours. It's not flashy or glamorous, but like batch scripting, it gets the job done. And sometimes, that's all that really matters.

Let's review some of the RegEx examples in C#.

---

### NOTE

For examples in PowerShell (and Batch), use this resource at my GitHub:

<https://github.com/mrnettek/Atomics/blob/main/RegEx-PowerShell.txt>



## Example Code

---

### RegEx: Basic Match, Simple String Search

---

```
using System;
using System.Text.RegularExpressions;

class Program
{
    static void Main()
    {
        // Define the input string to search within.
        string input = "Hello, RegEx User!";

        // Create a Regex object with the pattern "User".
        Regex regex = new Regex("User");

        // Perform the regular expression matching operation.
        Match match = regex.Match(input);

        // Display the original string to the user
        Console.WriteLine("Current Input string:");
        Console.WriteLine(input + "\n");

        // Check if the regex matches anything in the string.
        if (match.Success)
        {
            // If a match was found, inform the user.
            Console.WriteLine("Found 'User' in the Input string.");
        }
        else
        {
            // If no match was found, inform the user.
            Console.WriteLine("'User' not found.");
        }

        Console.ReadKey();
    }
}
```



## RegEx: Using '^' for String Start Matching

---

```
using System;
using System.Text.RegularExpressions;

class Program
{
    static void Main()
    {
        // Define the input string that we want to search within.
        string input = "Hello, RegEx User!";

        // Create a Regex object with the pattern "^Hello" to search.
        Regex regex = new Regex("^Hello");

        // Perform the regular expression matching operation (check Hello).
        Match match = regex.Match(input);

        // Display the original input string to the user.
        Console.WriteLine("Current Input string:");
        Console.WriteLine(input + "\n");

        // Check and display if the input starts with "Hello".
        Console.WriteLine(match.Success ? "Starts with 'Hello'" : "Does not
start with 'Hello'");
    }
}
```



## RegEx: Using '\$' for String End Matching

---

```
using System;
using System.Text.RegularExpressions;

class Program
{
    static void Main()
    {
        string input = "Goodbye, RegEx User!";
        Regex regex = new Regex("User!$");
        Match match = regex.Match(input);

        // Display the original input string to the user.
        Console.WriteLine("Current Input string:");
        Console.WriteLine(input + "\n");

        Console.WriteLine(match.Success ? "Found a digit." : "No digits
        found.");
        Console.ReadKey();
    }
}
```

## RegEx: Using '\d' to Match Digits

---

```
using System;
using System.Text.RegularExpressions;

class Program
{
    static void Main()
    {
        string input = "There are 32 computers.";
        Regex regex = new Regex(@"\d");
        Match match = regex.Match(input);

        // Display the original input string.
        Console.WriteLine("Current Input String:");
        Console.WriteLine(input + "\n");

        Console.WriteLine(match.Success ? "Found a digit." : "No digits
        found.");
    }
}
```



```
        Console.ReadKey();
    }
}
```

## RegEx: Using '\w' to Match Word Characters

---

```
using System;
using System.Text.RegularExpressions;

class Program
{
    static void Main()
    {
        // Define the input string to search for words.
        string input = "RegEx123!";

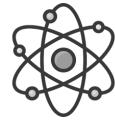
        // Create object Regex pattern @"\w+" to match word characters.
        Regex regex = new Regex(@"\w+");

        // Perform regular expression matching operation on the input string.
        Match match = regex.Match(input);

        // Display the original input string to the user.
        Console.WriteLine("Current Input String:");
        Console.WriteLine(input + "\n");

        // If match.Success is true, a word (matching the pattern) was found.
        Console.WriteLine(match.Success ? "Found a word!" : "No word found!");

        Console.ReadKey();
    }
}
```



## RegEx: Using '\s+' to Match Whitespace

---

```
using System;
using System.Text.RegularExpressions;

class Program
{
    static void Main()
    {
        // Define the input string to search for whitespace characters.
        string input = "Hello, RegEx User!";

        // Create object Regex pattern @"\"s" to match whitespace.
        Regex regex = new Regex(@"\s");

        // Perform RegEx matching operation for any whitespace.
        Match match = regex.Match(input);

        // Display Input String
        Console.WriteLine("Current Input String:");
        Console.WriteLine(input + "\n");

        // Check if the regex found any whitespace in the input string.
        // If match.Success, whitespace is found; otherwise not.
        Console.WriteLine(match.Success ? "Found whitespace!" : "No whitespace
        found!");

        Console.ReadKey();
    }
}
```



## RegEx: Using '\d+' to Return Phone Number

---

```
using System;
using System.Text.RegularExpressions;

class Program
{
    static void Main()
    {
        // Define the input string that contains a phone number.
        string input = "My phone number is 123-555-1234.";

        // Create object Regex pattern @"\d+" to match sequences of digits.
        // The pattern \d+ matches one or more digits (0-9).
        Regex regex = new Regex(@"\d+");

        // Perform the RegEx matching operation on the input string.
        MatchCollection matches = regex.Matches(input);

        // Display the original input string to the user.
        Console.WriteLine("Current Input String:");
        Console.WriteLine(input + "\n");

        // If digits are found, reconstruct the phone number.
        if (matches.Count >= 3) // Expect 3 digit groups valid phone number.
        {
            // Reconstruct the phone number as xxx-xxx-xxxx.
            string phoneNumber =
                $"{matches[0].Value}-{matches[1].Value}-{matches[2].Value}";

            // Display the full phone number
            Console.WriteLine($"Found phone number: {phoneNumber}");
        }
        else
        {
            // Not Found
            Console.WriteLine("No valid phone number found.");
        }

        Console.ReadKey();
    }
}
```



## RegEx: Using '\w' & '\w+' to Return Email Address

---

```
using System;
using System.Text.RegularExpressions;

class Program
{
    static void Main()
    {
        // Define the input string that contains an email address.
        string input = "Contact us at RegEx@example.com.";

        // Create object Regex pattern for matching an email address.
        // [\w\.-]+ matches word characters, periods, hyphens before @.
        // Pattern @ is used to match the @ symbol.
        // Pattern [\w\.-]+ match word char, periods, hyphens after the @.
        // Pattern \.\w+ matches period followed by word char: .com, .org.
        Regex regex = new Regex(@"[\w\.-]+\@[ \w\.-]+\.\w+");

        // Perform RegEx matching operation on the input string.
        Match match = regex.Match(input);

        // Display the original input string to the user.
        Console.WriteLine("Current Input String:");
        Console.WriteLine(input + "\n");

        // Check if an email address was found.
        if (match.Success)
        {
            // Display the found email address.
            Console.WriteLine($"Found email address: {match.Value}");
        }
        else
        {
            // If no email address was found, display a message.
            Console.WriteLine("No valid email address found.");
        }

        Console.ReadKey();
    }
}
```



## RegEx: Using '\b\d{2}\d{2}\d{4}\b' to Match Date

---

This example uses the regular expression `\b\d{2}\d{2}\d{4}\b` to match dates in the format `MM/DD/YYYY`. It ensures the date is properly formatted with two digits for the month, day, and four digits for the year, surrounded by word boundaries to avoid partial matches.

```
using System;
using System.Text.RegularExpressions;

class Program
{
    static void Main()
    {
        // Define the input string to search within.
        string input = "The event is on 12/25/2025.";

        // Example: Match a Date Format (MM/DD/YYYY)
        string datePattern = @"\b\d{2}\d{2}\d{4}\b"; // Matches dates
        Match dateMatch = Regex.Match(input, datePattern);

        // Check if the regex matches a date in the string.
        if (dateMatch.Success)
        {
            Console.WriteLine("\nFound a date in the Input string: " +
                dateMatch.Value);
        }
        else
        {
            Console.WriteLine("\nNo date found.");
        }

        Console.ReadKey();
    }
}
```



## RegEx: Using 'https?://[^\\s]+' to Match a URL

---

This example uses the regular expression `https?://[^\\s]+` to match URLs that start with "http" or "https". It captures the full URL by matching any characters that follow the protocol, stopping at any whitespace.

```
using System;
using System.Text.RegularExpressions;

class Program
{
    static void Main()
    {
        // Define the input string to search within.
        string input = "Visit us at https://www.google.com";

        // Example: Match a URL
        string urlPattern = @"https?://[^\\s]+"; // Matches URLs
        Match urlMatch = Regex.Match(input, urlPattern);

        // Check if the regex matches a URL in the string.
        if (urlMatch.Success)
        {
            Console.WriteLine("\nFound a URL in the Input string: " +
                urlMatch.Value);
        }
        else
        {
            Console.WriteLine("\nNo URL found.");
        }

        Console.ReadKey();
    }
}
```



## 23. What is the System Account?

---

The Windows **System Account**, often referred to as **LocalSystem** or just **System**, is a built-in account that operates with **unrestricted access** to nearly all resources on a machine. This account is designed to handle background services, critical operating system components, and other low-level tasks that require the highest possible privilege level. Most desktop management platforms will run processes in the System Account.

Unlike regular user accounts or even Administrator accounts, the System Account is unique in both its capabilities and its purpose:

- **Unmatched Privileges:** The System Account has full access to the operating system and can bypass many security restrictions that would limit other accounts. This allows it to perform actions critical to system functionality, such as modifying protected system files, managing device drivers, or interacting with the kernel.
- **Non-Interactive Nature:** It is not associated with a person and does not provide a standard login session. Instead, it runs in the background, performing tasks on behalf of the operating system and other services without direct user intervention.
- **Service Context:** Many Windows services, especially those integral to system stability and security, run under the System Account. This ensures they can operate without the limitations imposed on other accounts.
- **Minimal Environmental Variables:** The System Account operates with a pared-down set of environmental variables compared to Administrator or standard user accounts. Paths like %USERPROFILE% or %APPDATA%, which are essential in a user context, may not exist or will point to locations unrelated to individual users.
- **Registry Hive Access:** Instead of using HKEY\_CURRENT\_USER (HKCU), the System Account often references HKEY\_USERS\DEFAULT or runs without a user-specific registry hive entirely. This distinction is critical for scripts or applications that rely on user-specific settings or configurations.



- **Power and Risks:** While its unrestricted nature makes it indispensable for system processes, it also makes the System Account extremely sensitive. Any misuse—intentional or accidental—can compromise the entire machine, bypass security measures, or disrupt essential operations.

By design, the System Account is an essential part of the Windows ecosystem, enabling seamless operation of processes that need the highest level of trust and authority. Understanding its role and testing your code or configurations in this unique context is critical to ensuring reliability and security in enterprise environments.

## Leverage System Simulation

Use tools like the Sysinternals **PsExec** with the **-s** or other “run-as-system” utilities to mimic the System context (I’m a Sysinternal tools advocate). These tools help identify issues early, during the development and testing stages.

### Limit Privilege Usage

The System Account is immensely powerful. Use that power responsibly:

- Perform only tasks requiring system-level privileges.
- Enforce tight control over who can deploy or manage scripts running as System.

### Explicitly Handle Contexts

If your script handles both user-level and system-level tasks, clearly distinguish between them. This might involve:

- Redirecting file paths.
- Adjusting registry access.

Dynamically determining environment variables.



## Expect Missing Variables and Paths

Don't assume that user directories or registry keys exist. Handle missing paths gracefully by:

- Using **HKLM** for machine-wide settings.
- Storing resources in neutral locations.

## Build Robust Logging

Since debugging the System context lacks the benefit of interactive sessions, logging is critical. Log everything from registry access to environmental variables to quickly pinpoint failures.

## Document and Standardize

Establish clear standards for testing code intended to run under System. Document it thoroughly, and educate your team—many organizations still overlook this critical step.

## Testing in the System Account

Testing under the **System Account** is really non-negotiable for any enterprise-level solution. Ignoring this step introduces significant risks—scripts can fail silently, misbehave, or expose your organization to security threats.

Testing in the System Account is more than a checkbox; it's a safeguard against costly production issues and potential security breaches. Don't leave it to chance. Take the time to test properly, and your deployments will be rock-solid.

## PsExec Example

PsExec64.exe is an indispensable utility that empowers you to run processes under the System Account, making it essential for testing scripts, applications, and configurations in their intended environment. Among the tools I rely on, this one stands out as the most critical and versatile in my arsenal.



Now, it's time for you to test. Here are the steps to get you into the System Account:

### **Step 1 | Download PsExec64.exe**

Get the tool from Microsoft's Sysinternals Suite webpage. Don't trust any other website for the Sysinternal tools.

Extract it to a folder on your system. I recommend C:\Windows\System32, so you no longer have to deal with remembering where it lives.

### **Step 2 | Open Elevated Command Prompt**

**Type:** PsExec64.exe -s -i cmd.exe

### **Step 3 | Verify Security Context**

**Type:** Whoami

```
| output | nt authority\system
```

### **Step 4 | Test Command**

**Type:** echo %username%

```
| output | NameOfComputer$ --- It won't be your username, as it normally would.
```

### **Step 5 | Test Batch Script**

```
C:\Batch\script.cmd
```

### **Step 6 | Test PowerShell Script**

```
powershell.exe -ExecutionPolicy Bypass -File "C:\Path\To\YourScript.ps1"
```

While you're in the System Account, monitor script behavior, registry paths, environment variables, and other commands. Verify results and impact. And, be proud that you now understand what the System Account is and what it can do for you.



## 24. Formatted Timestamps

---

### What Are Formatted Timestamps?

A formatted timestamp is a structured representation of date and time, typically used in programming and scripting to maintain consistency, readability, and easy parsing. These timestamps often follow a standardized format like:

YYYYMMDD_HHMM	(Ex: 20250207_1530) – (My favorite format)
YYYY-MM-DD HH:MM:SS	(Ex: 2025-02-07 15:30:45)

What can you do with formatted timestamps?

- **Log Entries** – Use timestamps to create unique log entries, to track when processes and tasks were completed.
- **Log File Naming** – Use timestamps to create unique log file names, preventing overwrites.
- **Sorting and Organization** – Store timestamps in filenames or database records to enable easy chronological sorting.
- **Backup and Versioning** – Append timestamps to backup files or folder names for easy tracking of versions.
- **Naming Temporary Files** – Create temporary files with unique names to prevent conflicts in multi-user or multi-process environments.
- **Automated Task Scheduling** – Use timestamps to trigger scripts or automate tasks at specific times.
- **Time-Based Comparisons** – Compare timestamps to determine the age of files, logs, or database records.



## Format Examples

---

### Log File Naming

log\_20250207\_1530.txt

### Backup and Versioning

backup\_20250207\_1530.zip

### Audit and Debugging in Logs

[2025-02-07 15:45:30] Script executed successfully.

### Generating Unique Identifiers

user123\_20250207\_1530

### Sorting and Organization

invoice\_20250207\_1530.pdf

### Automated Task Scheduling

A script that runs every hour logs:

[2025-02-07 16:00:00] Task started.

### Archiving Old Data

C:\Archives\report\_20250101\_1200.xlsx

### Time-Based Comparisons

Identifying files older than 30 days using timestamps.

### Naming Temporary Files

tempfile\_20250207\_1530.tmp

### Timestamp-Based Reports

Sales\_report\_20250207\_1530.csv



## Example Code

---

### Batch Script

```
@ECHO OFF
:: Get Current Date
FOR /F "tokens=2 delims= " %%A IN ('DATE /T') DO SET "currDate=%%A"

:: Rearrange the date components (year, month, day) to create a consistent
timestamp.
FOR /F "tokens=1-3 delims=/-. " %%A IN ("%currDate%") DO SET
"timestamp_date=%%C%%A%%B"

:: Get Current Time
FOR /F "tokens=1-2 delims=:" %%A IN ('TIME /T') DO SET "currTime=%%A%%B"

:: Parse the time format to remove any trailing AM/PM.
FOR /F "tokens=1-2 delims= " %%A IN ("%currTime%") DO SET "timestamp_time=%%A"
rem %%B

:: Combine the formatted date and time.
SET "Timestamp=%timestamp_date%_%timestamp_time%"

:: Create Log File Name And Timestamp Entry
ECHO [%Timestamp%] Executed successfully. >"C:\Batch\%Timestamp_Log.txt"
ECHO Done!
PAUSE
```

### PowerShell Script

```
# Get the current timestamp in the format YYYYMMDD_HHMM
$timestamp = Get-Date -Format "yyyyMMdd_HHMM" # yyyyMMdd_HHMMss

# Define log file name with timestamp
$logFile = "C:\PowerShell\$timestamp`_log.txt"

# Write log entry
"[${timestamp}] Executed successfully." | Out-File -FilePath $LogFile -Append

# Output confirmation
Write-Host "Log file created: $LogFile"
```



## 25. Windows Commands & Executables

---

To be used in creating your scripted masterpieces.

### Common Windows Commands

---

These are built directly into **cmd.exe** and do not require external files to function:

#### File and Directory Management

- **DIR** - Lists directory contents.
- **CD** or **CHDIR** - Changes the current directory.
- **MD** or **MKDIR** - Creates a new directory.
- **RD** or **RMDIR** - Removes a directory.
- **DEL** or **ERASE** - Deletes files.
- **COPY** - Copies files.
- **MOVE** - Moves files or directories.
- **REN** or **RENAME** - Renames files.
- **TYPE** - Displays the contents of a file.
- **CLS** - Clears the command prompt screen.

#### System Information

- **ECHO** - Displays text or turns command-echoing on/off.
- **SET** - Manages environment variables.
- **VER** - Displays the Windows version.
- **VOL** - Displays the volume label of a drive.
- **PATH** - Displays or sets the search path for executables.
- **PROMPT** - Changes the command prompt appearance.



## Flow Control

- **IF** - Performs conditional processing.
- **FOR** - Iterates over files, strings, or ranges.
- **GOTO** - Jumps to a labeled section in the script.
- **CALL** - Calls another batch file or a label in the same batch file.
- **PAUSE** - Pauses execution until a key is pressed.
- **EXIT** - Exits the script or command prompt.
- **SHIFT** - Changes the position of batch file parameters.

## Network and System Utilities

- **NET** - Manages users, shares, and network settings.
- **PING** - Checks network connectivity.
- **TRACERT** - Traces the route to a network address.
- **TITLE** - Sets the window title of the command prompt.
- **COLOR** - Changes the console Foreground and background colors.



## Common Windows Executables

---

These are standalone programs located in system directories like `C:\Windows\System32`.

### File and Directory Utilities

- **ATTRIB.EXE** - Modifies file attributes.
- **XCOPY.EXE** - Copies files and directories, including subdirectories.
- **ROBOCOPY.EXE** - Robust copy utility for files and directories.
- **DELPROF.EXE** - Deletes user profiles.

### System and Network Utilities

- **IPCONFIG.EXE** - Displays network configuration.
- **NSLOOKUP.EXE** - Queries DNS.
- **TASKLIST.EXE** - Displays running processes.
- **TASKKILL.EXE** - Terminates processes.
- **SC.EXE** - Manages Windows services.
- **WMIC.EXE** - Provides WMI management capabilities.

### System Management

- **CHKDSK.EXE** - Checks and repairs disk errors.
- **DISKPART.EXE** - Manages disk partitions.
- **SFC.EXE** - Scans and repairs system files.
- **BOOTREC.EXE** - Repairs boot configuration data.
- **BCDEDIT.EXE** - Edits boot configuration data.

### Compression and Archiving

- **COMPACT.EXE** - Compresses files on NTFS drives.
- **EXPAND.EXE** - Expands compressed files.
- **MAKECAB.EXE** - Creates compressed cabinet (.CAB) files.



## Debugging and Diagnostics

- **DEBUG.EXE** - Used for low-level debugging (legacy tool).
- **CERTUTIL.EXE** - Manages certificates.
- **EVENTVWR.EXE** - Opens the Event Viewer.

## How to Distinguish Commands from Executables

---

1. **Commands:** Run directly in **cmd.exe**. They are case-insensitive and do not require an .exe extension.  
**Example:** DIR, COPY, IF
2. **Executables:** Require the file to exist in a system path or directory.  
**Example:** ping, ipconfig, notepad

### NOTE

---

<https://learn.microsoft.com/en-us/windows-server/administration/windows-commands/windows-commands>



## 26. Sysinternals Tools by Mark Russinovich

---

**Expand the power of your scripts by learning how to use other tools.**

The Sysinternals Tools, developed by Mark Russinovich, are an indispensable suite of utilities for IT professionals, system administrators, and power users seeking unparalleled insight into Windows systems. These tools empower users to troubleshoot complex problems, optimize performance, secure their environments with precision and reliability, and increase the power of scripts and automation. Covering a wide array of functions, Sysinternals provides capabilities to monitor processes in real-time, manage file permissions, analyze system performance, debug application crashes, and securely delete sensitive data.

From Process Explorer, a feature-rich alternative to Task Manager, to Autoruns, which gives a comprehensive view of startup programs and services, each tool is designed to uncover details hidden from standard system utilities. Tools like TCPView and Process Monitor make tracking network connections and system activities a breeze, while Sysmon provides advanced monitoring for security professionals. Whether you're diagnosing an elusive software issue, investigating network activity, or auditing system configurations, Sysinternals equips you with the deep-level insights needed to stay ahead.

For anyone serious about understanding what happens under the hood of Windows, the Sysinternals suite isn't just helpful—it's essential. Accessible, lightweight, and incredibly powerful, these tools have become a gold standard in system management and troubleshooting.

**Check out these books by Russinovich:**

*Troubleshooting with the Windows Sysinternals Tools*

*Windows Internals: System architecture, processes, threads, memory management, and more*



Here's an overview of the Sysinternal utilities (<https://learn.microsoft.com/en-us/sysinternals>)

## File and Disk Utilities

- **AccessChk** - Displays permissions of files, folders, registry keys, and more.
- **AccessEnum** - Shows who has access to directories and files.
- **Contig** - Defragments individual files for improved performance.
- **Disk2vhd** - Creates VHD (Virtual Hard Disk) images of physical drives.
- **Du (Disk Usage)** - Shows disk space usage by folder.
- **Handle** - Displays open file handles and the processes using them.
- **NTFSInfo** - Shows NTFS volume and file system details.
- **Streams** - Reveals alternate data streams in NTFS files.
- **Sync** - Flushes file system buffers to disk.

## Process Utilities

- **Autoruns** - Displays programs configured to run at system startup.
- **Procmon (Process Monitor)** - Monitors real-time file, registry, and process/thread activity.
- **Procexp (Process Explorer)** - Advanced Task Manager with detailed process insights.
- **PsExec** - Executes processes on remote systems.
- **PsKill** - Terminates processes on local or remote systems.
- **PsList** - Lists detailed information about processes.
- **PsSuspend** - Suspends processes temporarily.



## Network Utilities

- **TCPView** - Shows active TCP/UDP connections.
- **PsPing** - Measures network latency and bandwidth.
- **Whois** - Queries domain registration information.
- **BgInfo** - Displays system information on the desktop.
- **PortMon** - Monitors serial and parallel port activity.

## System Monitoring Tools

- **Autoruns** - Tracks all auto-start locations for startup programs.
- **Process Explorer** - Provides detailed process analysis and resource usage.
- **VMMMap** - Visualizes memory usage by processes.
- **RAMMap** - Analyzes physical memory usage.
- **LiveKd** - Executes kernel debuggers against live systems or crash dumps.

## Security and Permissions

- **Sigcheck** - Verifies digital signatures of files.
- **PsLoggedOn** - Shows users logged on locally or via resource access.
- **PsPassword** - Resets local user account passwords.
- **AccessChk** - Checks effective permissions for users or groups.
- **SDelete (Secure Delete)** - Permanently deletes files securely.



## Registry Utilities

- **RegDelNull** - Deletes registry keys with embedded null characters.
- **RegMon** - Monitors real-time registry activity (merged into Procmon).

## Diagnostics Tools

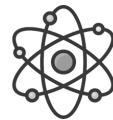
- **BlueScreen** - Simulates a Blue Screen of Death for training or testing.
- **DebugView** - Captures real-time debug output on local or remote machines.
- **CoreInfo** - Shows CPU features and NUMA node information.
- **ClockRes** - Displays system timer resolution.
- **AdExplorer** - Views and searches Active Directory information.

## Virtualization Tools

- **Desktops** - Creates virtual desktops for better workspace organization.
- **ZoomIt** - Allows screen zooming and annotation.
- **BgInfo** - Adds system information to the desktop background.

## Miscellaneous Utilities

- **PendMoves** - Displays file operations pending after a reboot.
- **MoveFile** - Renames or deletes files on the next reboot.
- **Strings** - Extracts printable text from binary files.
- **NotMyFault** - Tests and diagnoses BSOD issues by generating crashes.



## 27. Atomic Series GitHub

---

### Help Files for Further Reading

These help files are located at: <https://github.com/mrnettek/Atomics>

Name	Description
<b>BitsAdmin</b>	Used for managing the Background Intelligent Transfer Service (BITS), which is used for transferring large files over networks with minimal disruption.
<b>Boolean Logic</b>	Boolean logic deals with expressions that evaluate to one of two values: \$true or \$false.
<b>Cacls</b>	Used to display or modify the Access Control Lists (ACLs) for files and folders.
<b>CertUtil</b>	Used to manage certificates, certificate authorities (CA), and related configurations, as well as back up and restore CA components.
<b>Color</b>	Used to set the default console foreground and background colors in the Command Prompt window. This allows you to customize the appearance of the console for better readability or to match personal preferences.
<b>Copy</b>	Used to copy one or more files from one location to another. It supports a variety of options for handling different types of files, merging multiple files, and customizing the behavior of the copy operation.
<b>Curl</b>	Used for transferring data from or to a server, supporting various protocols such as HTTP, HTTPS, FTP, FTPS, SCP, SFTP, TFTP, DICT, TELNET, LDAP, and FILE.
<b>Date</b>	Used to display or set the system date.
<b>Del</b>	Used to delete one or more files.
<b>Dism</b>	Used to service Windows images, including the installation, configuration, and updating of features, drivers, and packages. It can be used to manage both online and offline Windows images.



<b>DsRegCmd</b>	Used to display the device join status and manage hybrid Azure AD join operations. It is used to query, join, leave, or refresh a device's Azure AD status, among other tasks.
<b>Find</b>	Used to search for a text string in a file and display all the lines where it is found. Below is a brief guide with syntax and examples.
<b>FindStr</b>	Used to search for a string in files or input and display lines containing the string. It provides more advanced searching capabilities compared to FIND, such as support for regular expressions and multiple search strings.
<b>For Loop</b>	Used to perform a command several times, based on a set of data. It is commonly used to iterate over files, folders, numbers, or strings, and can be utilized in various ways depending on the syntax.
<b>For /D</b>	Used to perform an action on directories (folders) that match a specified pattern. It is similar to the regular FOR command but operates specifically on directories and supports wildcard patterns.
<b>For /F</b>	Used to iterate through a file, a command's output, or a string by parsing its content. This is especially useful for processing text files, filtering data, or extracting specific tokens (subsections of a line).
<b>For /L</b>	Used to loop over a specific range of integer values. You can use a start value, step value, and an end value to generate a sequence of numbers, and then perform a specified command (or set of commands) for each number in that sequence.
<b>ForFiles</b>	Used to select files or sets of files based on various criteria and execute a command for each selected file. It is commonly used for batch processing tasks, such as file management, renaming, or applying commands to groups of files.
<b>Goto</b>	Used to jump to a specific label within the script. Labels are defined within the script and serve as points where the batch file execution can jump to.
<b>iCacls</b>	Used to display, modify, and manage Access Control Lists (ACLs) for files and folders. It resolves issues that occurred when using the older CACLS and XCACLS utilities. iCACLS allows administrators to manage file permissions with greater flexibility and security.
<b>If Condition</b>	Used to execute a command or a block of commands if a given condition evaluates to true. It is a fundamental part of decision-making in scripts, allowing for conditional execution based on file checks, variable comparisons, or system properties.
<b>If-Else Condition</b>	Used to execute different commands based on whether a condition is true or false. It's used for decision-making in scripts, enabling conditional execution of code blocks. Batch scripts often use IF for checking file existence, variable values, or



	system properties, and ELSE provides an alternative action if the condition is not met.
<b>IpConfig</b>	Used to configure and display the IP (Internet Protocol) configuration of the system. It is primarily used for managing the IP address settings and troubleshooting network connectivity issues.
<b>Manage-Bde</b>	Used to configure BitLocker Drive Encryption on disk volumes. It provides a set of commands for managing BitLocker encryption, including enabling or disabling encryption, managing recovery keys, and configuring automatic unlocking for data volumes.
<b>Move</b>	Used to move a file or folder from one location to another or to rename it. It can also be used to move multiple files using wildcards.
<b>Net Accounts</b>	Used to manage user accounts and groups on both local and remote systems, set password policies, and configure logon restrictions.
<b>Net Session</b>	Used to manage open files and user sessions on a computer or network. It provides functionality for viewing and managing user sessions, as well as managing open files on a shared server.
<b>Net Share</b>	Used to manage file and printer shares on a local or remote computer. It allows you to create, modify, and delete shares, set access permissions, and configure cache settings for shared folders or devices.
<b>Net Start</b>	Used to start, stop, pause, or resume services that are listed in Control Panel > Services.
<b>NetSh</b>	Used to configure network interfaces, Windows Firewall, routing, remote access, and other network-related settings. It provides a wide range of commands for managing networking features in Windows.
<b>NtRights</b>	Used to edit user account privileges in a Windows system. It can be used to grant or revoke user rights such as logon rights, system administrator privileges, and other service or system-level permissions.
<b>Ping</b>	Used to test the network connection between your computer and another host. It sends an ICMP Echo Request message to the destination host and waits for a reply, showing the round-trip time taken for the message to reach the destination and return. This can help in troubleshooting network connectivity issues.
<b>PopD</b>	Used to change the current directory back to the path most recently stored by the PUSHD command. It also removes any temporary drive mappings created by PUSHD. This command allows you to navigate back through directories in a "stack" (LIFO—last in, first out) order, making it easier to return to previous directories.



<b>PowerCfg</b>	Used to control and configure power settings, such as sleep, hibernate, and standby modes. It allows users to modify, query, and delete power schemes, as well as manage device wake-up settings.
<b>PushD</b>	Used to change the current directory or folder while storing the previous path so that it can be easily returned to using the POPD command. This is useful for navigating between directories in batch scripts or in interactive command-line use.
<b>QUser</b>	Used to display information about user sessions on a Terminal Server or a Remote Desktop Session Host (RD Session Host) server. This command provides details about user sessions such as their state, idle time, and logon time.
<b>QWinsta</b>	Used to display information about user sessions on a Terminal Server or Remote Desktop Session Host (RD Session Host) server. This command provides details about active, idle, and disconnected sessions, session IDs, states, and more.
<b>Rd</b>	Used to delete an empty directory or a directory along with its contents. This command can also be used to remove entire directory trees and folders.
<b>Reg</b>	Used to read, set, delete, and manage the Windows registry.
<b>RoboCopy</b>	Used to copy files and entire directory trees from one location to another. It offers advanced features like multi-threaded copying, filtering options, and support for resuming interrupted transfers.
<b>RegEx</b>	Is a sequence of characters that defines a search pattern used for matching, locating, and manipulating text in strings based on specific rules.
<b>Sc</b>	Used to manage Windows services. It allows users to create, start, stop, query, or delete services, and perform other service-related operations such as querying a service's status, configuration, or dependencies.
<b>SchTasks</b>	Used to create, modify, delete, or query scheduled tasks in Windows. It can be run on local or remote systems, and provides detailed control over scheduling tasks based on specific triggers and conditions.
<b>SetAcl</b>	Used to modify security descriptors for files, folders, registry keys, services, printers, and shares on local or remote systems. It allows for managing access control lists (ACLs), setting ownership, and configuring advanced permissions for users and groups.
<b>SetX</b>	Used to set environment variables permanently in the Windows operating system. It can set environment variables for the current user (HKCU) or for the machine (HKLM), and changes made are written to the registry.



<b>SignTool</b>	Used to digitally sign files, verify digital signatures, and timestamp files. This tool is important for ensuring that files are authentic and haven't been tampered with, especially for software distribution, updates, and other security-sensitive activities.
<b>Sort</b>	Used to sort the contents of a file or input, either from the command line or from a file redirected into the command. It will output the sorted lines in ascending order by default.
<b>TakeOwn</b>	Used to take ownership of a file or folder. This command is typically used when a user needs to assume control over a file or folder that they don't currently have ownership of, allowing them to change permissions or delete the object.
<b>Taskkill</b>	Used to terminate one or more processes, either by process ID (PID) or by image name (process name). It can be used to stop processes on both local and remote systems.
<b>Tasklist</b>	Used to provide a list of currently running processes on a local or remote machine. It displays information such as the process name, process ID (PID), memory usage, and more. Below is a brief guide with syntax and examples.
<b>Time</b>	Used to display or set the system time. Below is a brief guide with syntax and examples.
<b>Timeout</b>	Used to pause the execution of a script or batch file for a specified duration. It can be used to create delays, allowing for time-based operations, or to pause until a user interaction is received.
<b>Type</b>	Used to display the contents of one or more text files in the command prompt. It is commonly used for viewing file contents, and in conjunction with redirection, can also be used for file manipulation.
<b>TzUtil</b>	Used to display or set the time zone for the current user. It can display the current time zone, list all available time zones, or change the time zone to a specified value.
<b>Ver</b>	Used to display the current version of the Windows Operating System. It outputs the major and minor version numbers, as well as the build number.
<b>WEvtUtil</b>	Used to manage Windows Event Logs and Event Publishers. You can use it to retrieve information about event logs, archive logs, clear logs, query events, and manage event publishers and manifests.
<b>Wmic</b>	Used to retrieve a wide range of information about local or remote computers, manage system configurations, and perform administrative tasks. It interacts with the Windows Management Instrumentation (WMI) to query system components, such as disk drives, printers, network interfaces, and more.



<b>Xcopy</b>	Used to copy files and directory trees to another location. It provides more advanced options than the COPY command, including the ability to copy directories, subdirectories, and various attributes.
--------------	---

See compiled list here:

<https://github.com/mrnettek/Atomics/blob/main/Commands-Tools.txt>



**END OF LINE**

