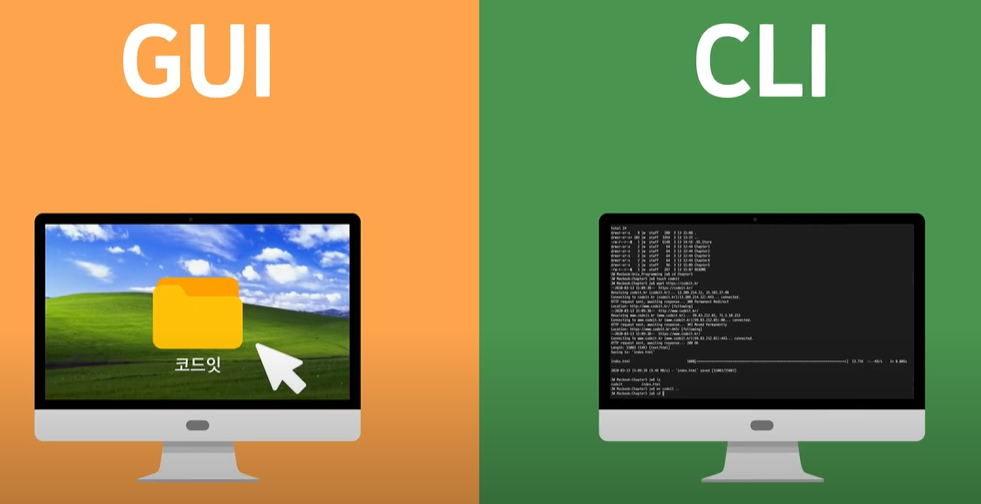
WINDOWS COMMANDLINE PART 1

The Command Line's Power:

The command line remains a **powerful tool for system administrators** and users, despite the GUI taking center stage. While the GUI can be convenient, it doesn’t give access to everything you need, and relying on it for certain tasks can be inefficient and error-prone.



Automation and Efficiency:

The command line allows for **automation of tasks**, saving time and reducing the chances of mistakes. For instance, once you learn the right commands, you can execute multiple tasks quickly without repeating actions manually as you would in the GUI.



Windows PowerShell and the Command Line:

Though **PowerShell is increasingly being used**, the command line is still very much supported in Windows Vista. In fact, Vista introduced significant updates to the command line, and the book highlights these changes.



Why the Command Line is Better:

The **GUI often fails in searching for specific files or tasks**, especially when it comes to searching executables. Tools like the **FindStr** utility (explained in Chapter 4) offer high-speed searching that the GUI can’t match.



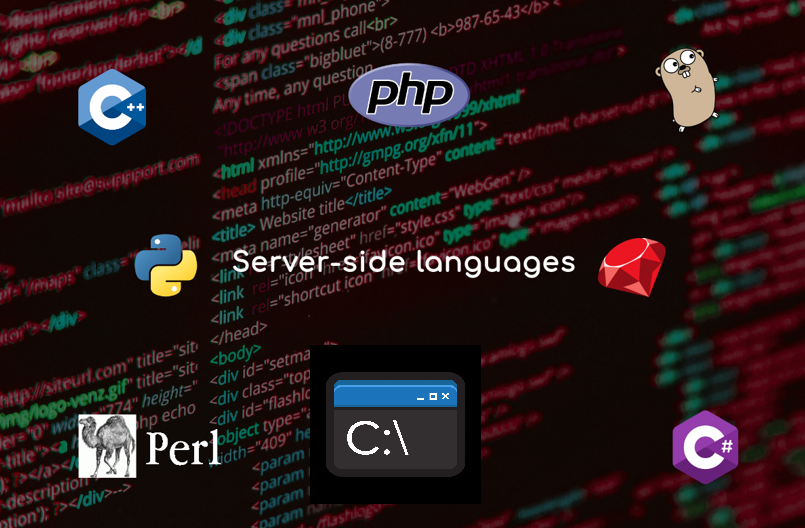
Detailed Information and Real-World Use:

The book doesn’t just teach basic command usage; it provides in-depth explanations, tips, and **real-world scenarios** to make the commands more useful. It’s designed for both beginners (with Chapter 1 for those new to the command line) and experienced users.



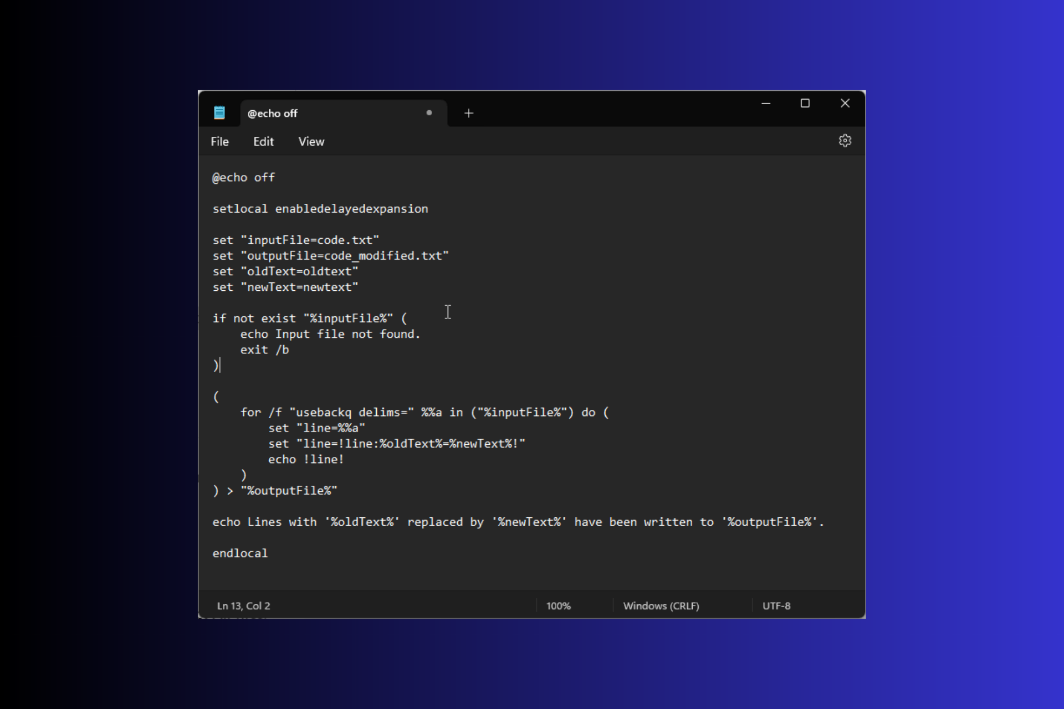
Real World Scenarios:

Each command and utility is backed by real-world stories, offering **practical examples** of how and why to use them, as well as potential pitfalls to watch out for.



Scripts and Batch Files:

The author provides scripts and batch files based on personal experience, which readers can use or adapt for their needs. These tools are presented as real-world, tested solutions.



THE BOOK AND WHAT YOU’LL LEARN

Broad Appeal (Chapters 1-5):

The initial chapters are designed for anyone who uses Windows, regardless of their technical expertise. These chapters will introduce powerful command-line tools that can simplify everyday tasks.



Basic Windows Knowledge Assumed:

While the early chapters are for everyone, the book assumes you have a foundational understanding of how to use Windows, such as navigating files and using basic input devices. It jumps directly into command-line usage without covering Windows basics.



Developer and Administrator Focus (Chapters 6-17):

These chapters delve into more advanced topics specifically for developers and system administrators. This includes commands for tasks not possible in the GUI, **accessing functions within DLL files**, automating tasks, using helpful third-party applications, and exploring the .NET Framework (like compiling JScript).



Looking Ahead to Vista and PowerShell (Chapters 18-19):

The final chapters provide a glimpse into the command-line experience in Windows Vista and **introduce Windows PowerShell**. PowerShell is presented as a significant advancement, offering more power, efficiency, and better security, similar to UNIX shells. It's also mentioned that PowerShell can be downloaded for older versions of Windows.



STANDARD WINDOWS UTILITIES

**🔧** Standard Windows Utilities — Chapter Breakdown

**Ch. 1**: Using the Command Line Effectively  
**Ch. 2**: Completing Data-Specific Tasks  
**Ch. 3**: Discovering the System Status  
**Ch. 4**: Locating Files & Other Resources  
**Ch. 5**: Securing & Monitoring a System  
**Ch. 6**: Developer & Low-Level Utilities

**📘** Chapter 1: Using the Command Line Effectively

**🧠 Why It Matters**

* Command line = power tool. GUI = pretty but limited.
* The command line lets you work faster, automate, and **make fewer mistakes**.
* Many critical system features aren’t in the GUI at all—only accessible from the CLI.

🗨️ *"It’s not outdated—it’s underground."*

**📉** Why People Dropped It

* GUI made computers more user-friendly—but it didn’t replace everything.
* Command line seems harder (syntax, memorization), but modern CLI is more scriptable and forgiving.
* Most people don't realize how much is **missing** from the GUI or how **many tasks** are easier from the CLI.

**🔄** Evolution of the Command Line

* DOS: full command line days.
* Windows: GUI took over, but the command line **never disappeared**.
* Vista: introduced **tons of new CLI tools**, despite the rise of PowerShell.

⚡ *This book covers ~280 command line utilities (standard + Vista-specific).*

**💡** What You Can Do with It

* Automate tasks with **batch files or scripts** (no need to remember commands every time).
* Run commands in background, on schedule, or without user input.
* Replace 3rd-party tools (especially sketchy search tools) with native ones like **FindStr** and **Dir**.

**🛠️** Tools & Concepts Introduced in Chapter 1

* **Viewing commands by purpose** (instead of memorizing alphabetically)
* **Internal vs External commands** (Internal = built into cmd.exe, e.g., dir, cd; External = separate .exe files)
* **Customizing the command window** (font, size, colors, scrollback, etc.)
* **Updating utilities via Microsoft Download Center**
* **Vista-specific command differences** (new parameters, tools)

**📌** Takeaways

* CLI is about **efficiency**, **power**, and **control**.
* Batch files/scripts save time and remove repetition.
* You already have a goldmine of tools on your system—this book helps you dig them out.

**📌** Understanding Why the Command Line Is So Important

**🖼️** A Little History

* **Windows was born** to:
  + Allow multitasking (DOS was one-app-at-a-time).
  + Be more *“user-friendly”* (no need to memorize commands).
* GUI succeeded in making interactive apps easier (word processors, dev tools).
* But many **noninteractive/background tasks** are still better handled with CLI.

**🛠️** When GUI Just Ain’t It

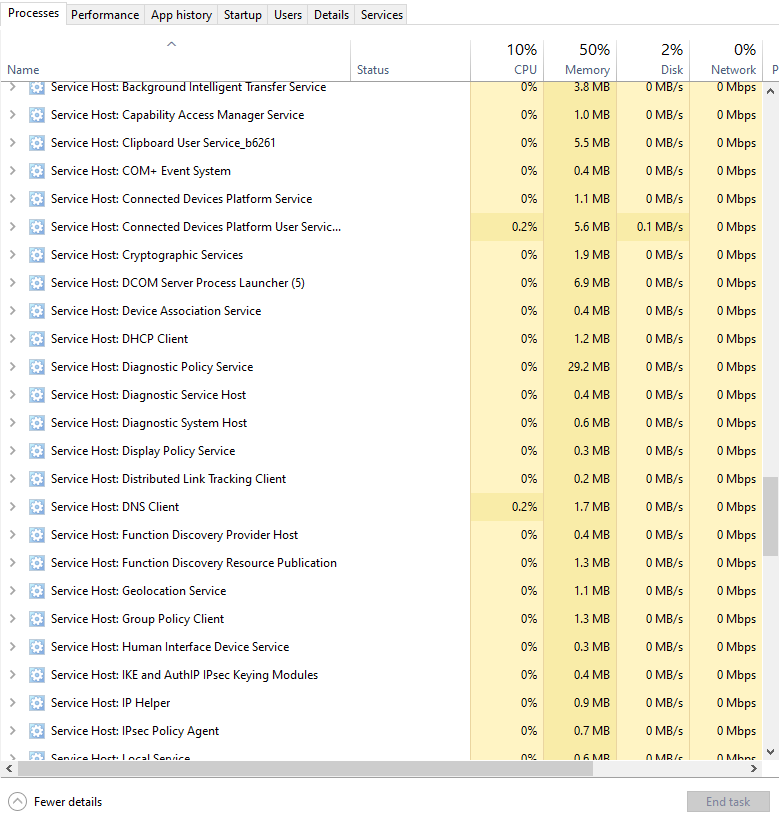
* Example: **Backups**.
  + Should run in background.
  + Need speed, accuracy, and repeatability.
  + GUI = not ideal (you’re using files, it slows down, and not easily automated).
  + CLI = perfect (no need for user input, can run scheduled).
* CLI > GUI for **automation**.

**🔍** Real Talk: Search is Broken in Explorer

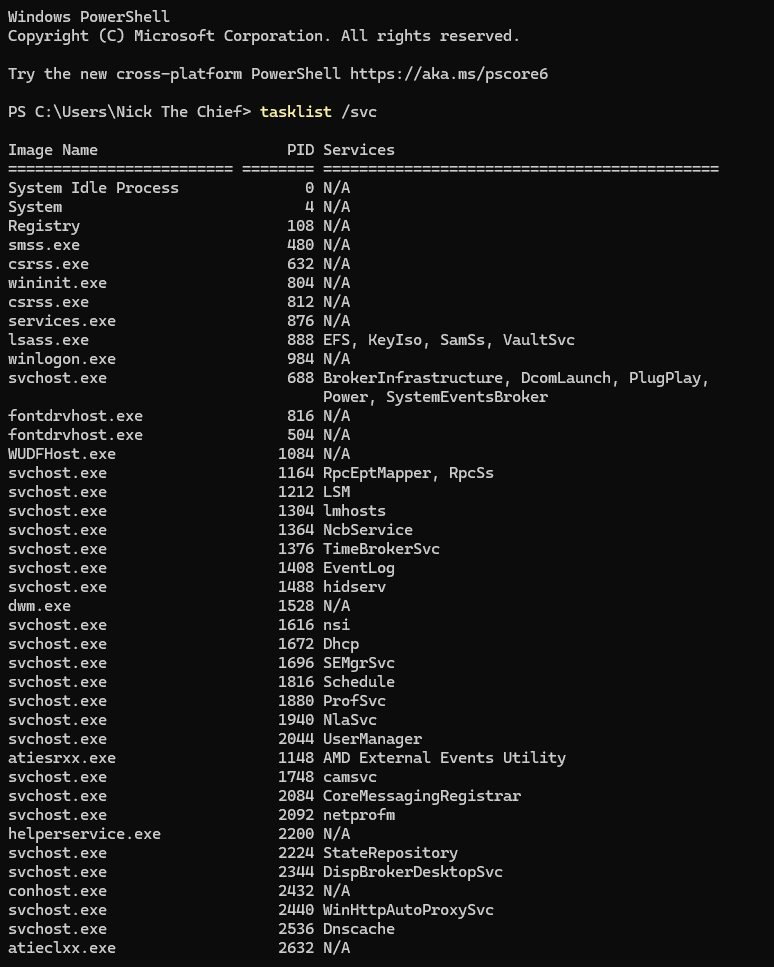
* GUI search is **slow**, **inaccurate**, and **doesn’t index** everything (even on Vista).
* Want to search inside binaries or random file types? GUI fails.
* **CLI tool like FindStr** crushes it.
  + Searches **inside any file**, including executables.
  + Fast, direct, and accurate.

**🕵️‍♂️** Security: What’s Actually Running?

* **GUI’s Task Manager:**
  + Limited view (SvcHost entries are vague).
  + Even Window’s “Services” tab helps *a little*.



* **CLI’s TaskList:**
  + Shows exact services running **inside** each SvcHost.
  + Reveals **PID** (Process ID) so you can trace what's really happening.

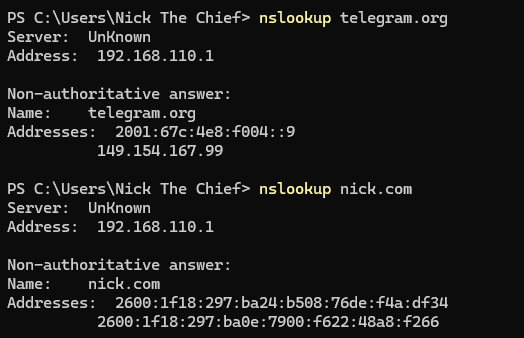


🧠 You can’t defend your system if you don’t know what it’s doing. CLI tells the truth, GUI hides it.

**🧠** CLI: Your X-Ray Goggles for Windows

**🌐** Network & Web Intelligence

* **DNS Who?** → Use CLI to resolve exact IP addresses behind friendly URLs.
  + Example: nslookup openai.com = no more guessing who’s spoofing your emails.
* **Slow Internet?** → Diagnose if it’s your Wi-Fi, your ISP, or a bad route on the web.
  + Tools like ping, tracert, and pathping tell the truth before you waste an hour on customer support.

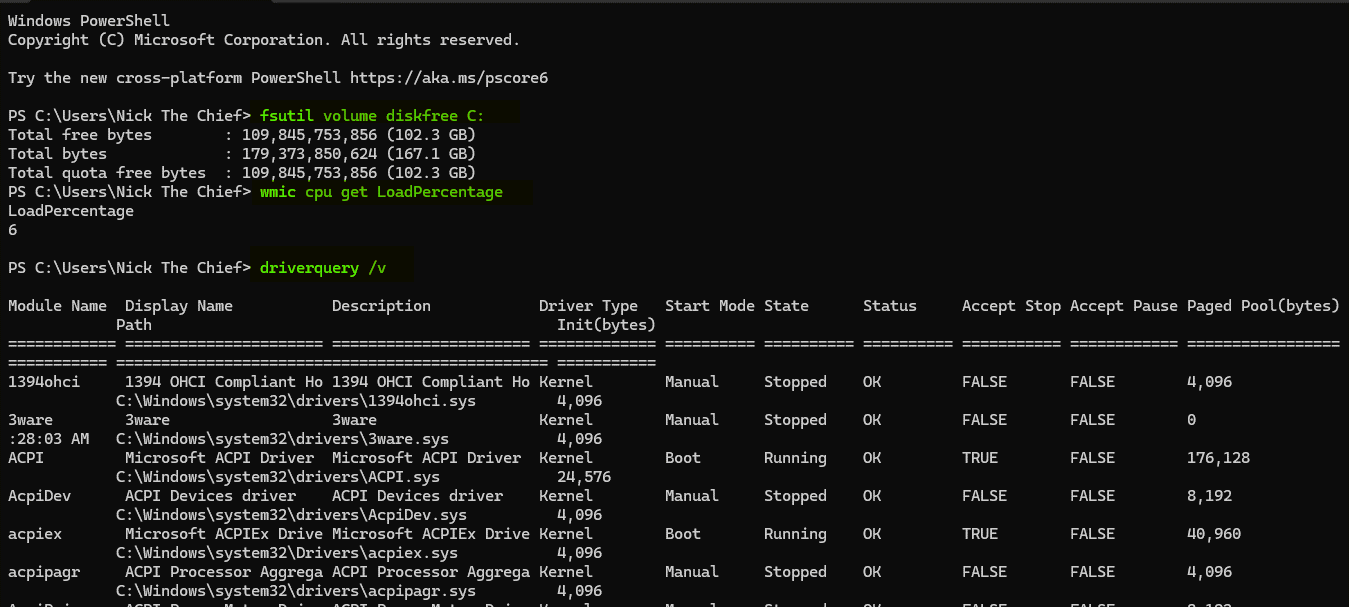


**🏠** Local + Network Resource Discovery

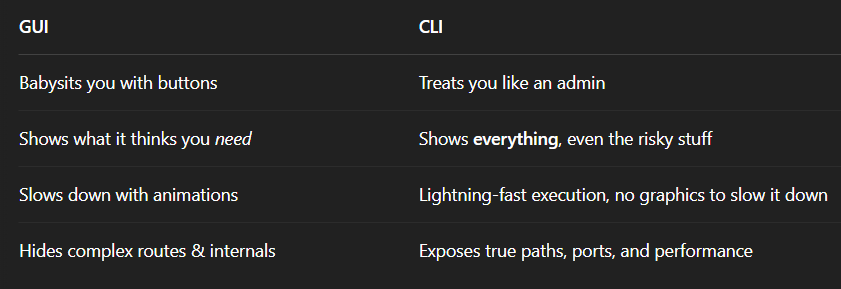
* CLI doesn’t just talk to the web—it **maps your local network** too.
  + netstat, ipconfig, nbtstat, arp, and friends help you find rogue devices, hidden services, and weird traffic.
  + Forget the GUI’s “Network” tab—it only shows happy paths. CLI shows **everything**, including what *shouldn’t* be there.

**💻** Deep System Visibility

* The command line sees stuff the GUI pretends doesn’t exist. You want actual disk usage? Real-time CPU info? Driver status? It's all there.
* CLI **doesn’t assume** what you’re trying to do—it just **does what you tell it**.



**🏴‍☠️** Why CLI Is Essential (Even If You Don't Use It Daily)



⚠️ Reality check:

If you’re serious about system security or performance, you **can’t trust** the GUI alone. *The CLI ain’t dead. It’s just quiet.* And like most quiet things in tech, it’s where the **power** lives. You don’t *have* to use it every day... but you should know it like a sidearm.

**🎯** Command Line = NOT That Deep (If You Know the Vibe)

**😴** What People Think the CLI Is:

“You just type a bunch of cryptic stuff, hope you don’t brick your machine, then stare at white text on a black screen like a hacker in a bad movie.”

Wrong. That’s the **myth**.



**🚪** CLI Is Easy If You Know the Secret Knock

You don’t need to memorize commands like it’s the SAT.  
You need **just one trick**:

🔑 **Add /? to *any* command to get instant help.**

**🤯 That’s it.**

* **tasklist /?**
* **xcopy /?**
* **netstat /?**
* **shutdown /?**

Almost *every* built-in Windows CLI tool responds to that magic switch and dumps a cheat sheet right in your terminal.

You’ll see all the **options** the command supports (scrollable if it's long).

**🧠** Why You Should Use Commands Directly First

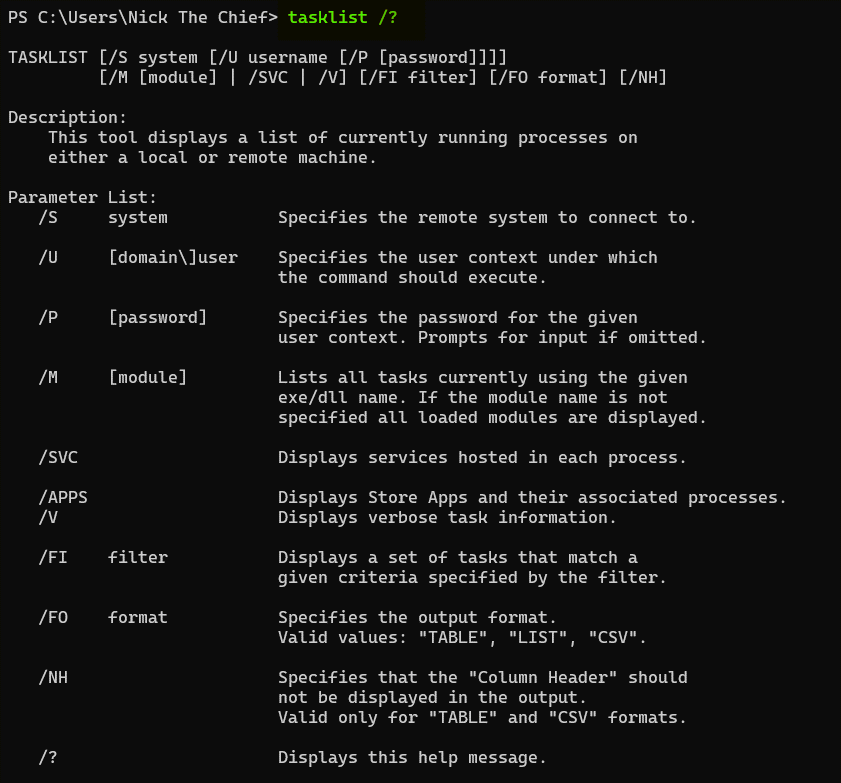
Yeah, scripting and batch files are cool...  
But before you automate anything, you’ve gotta understand:

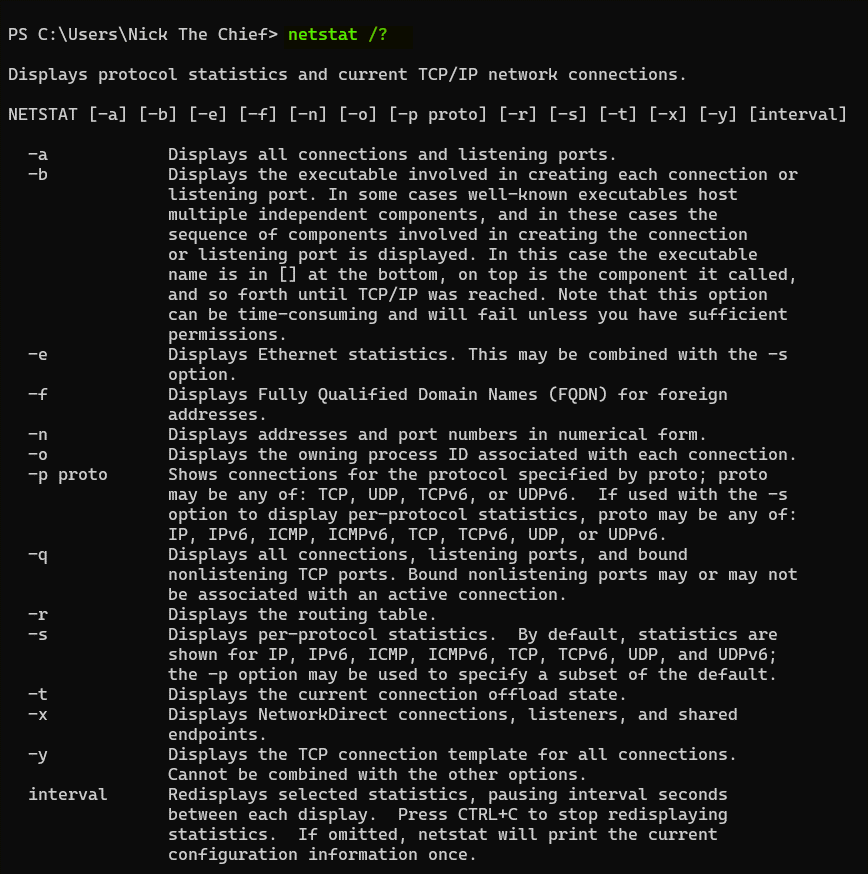
* What each command does
* What input it needs
* What output it gives

Once you *get* a few commands, stringing them together into automation is ✨child’s play✨.

**⚡** Real Talk

***If you know how to call a command with /? and read its output, you're already ahead of 90% of Windows users.*** Wanna keep the momentum going? Next up is **how to view commands by purpose**—so instead of guessing what tools exist, you’ll know where each one fits in the big picture (file ops, networking, diagnostics, etc.).





COMMANDLINE SWITCHES AND PIPES

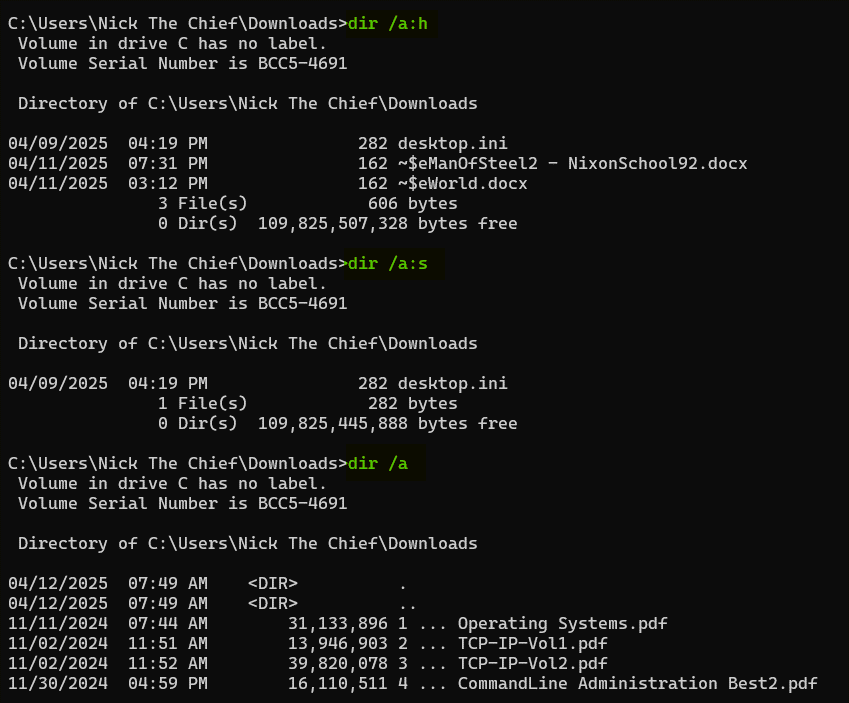
Think of a command-line switch as an **option** or a **modifier** that you add to a command to change how it behaves or what information it displays. It's like adding extra instructions to the main command.

**Switches** allow you to customize the command's output or its actions. They let you be more specific about what you want the command to do.

**Syntax:** As the book mentioned, switches usually start with a forward slash (/) or sometimes a dash (-) followed by a letter or a word.

Dir

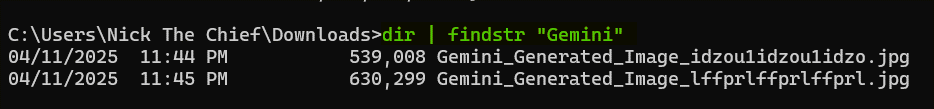
If you just type **dir** and press Enter, it will show a basic list. However, if you want to see more details, like the file size and the last modified date, you can use the **/W switch** (for "wide" format) or the **/A switch** (to display files with specific attributes). So, Dir /W would show a wider listing, and Dir /A:H would show hidden files.



Pipes (|):

A **pipe is a symbol (|)** used on the command line to **send the output of one command as the input to another command.** It's like creating a connection between two commands, where the first command's result becomes what the second command works with.

Let's say you want to see a list of all the files in a directory, but you only want to see the ones that contain the word "Gemini" in their name.



**Switches** modify the behavior of a single command. **Use / or -**

**Pipes** connect the output of one command to the input of another, allowing you to chain commands together for more complex operations. **Use |**

⚡ COMMAND LINE SYNTAX

Most Windows CLI commands follow this format:



Understanding Optional Parameters (Square Brackets [ ])

In command syntax, **square brackets**[ ] mean that the enclosed parameter is **optional**—you can include it or omit it.

Example:



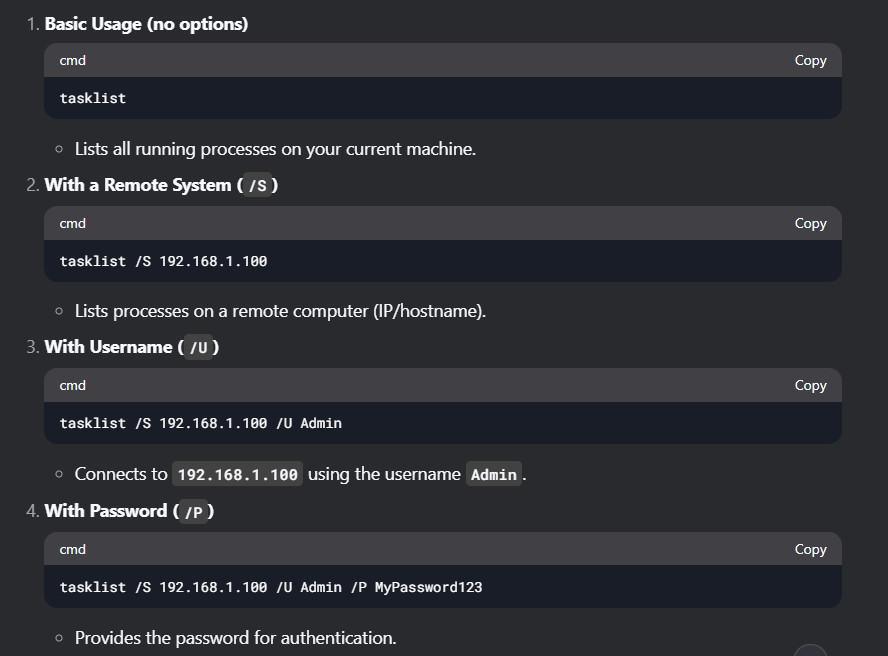
Means:

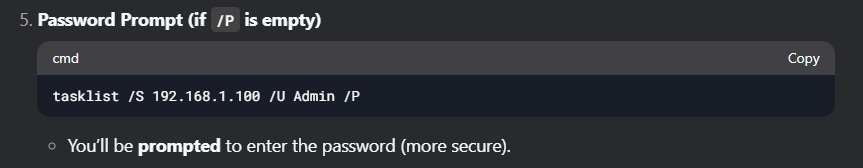
**You can run just tasklist with or without options.** Some might might misinterpret it as you write the command with the brackets, which is wrong, there’s nothing like writing the commands with brackets included. It means:

* Or add **/S** system (optional).
* If you use /S, you can optionally add **/U** username.
* If you use /U, you can optionally add **/P** password.

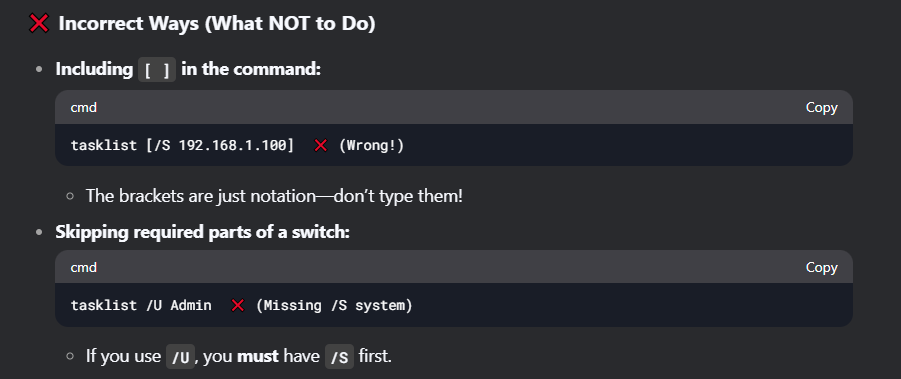
The square brackets ([ ]) are **just documentation syntax** — not something you actually write. They mean, *“This part is optional.”* They **do not** mean “copy me into your command.”

✅ Correct Ways to Run It:





You can do this with Virtual Machines which is a simpler way to explore. Let’s do it, I have some VMs ready.

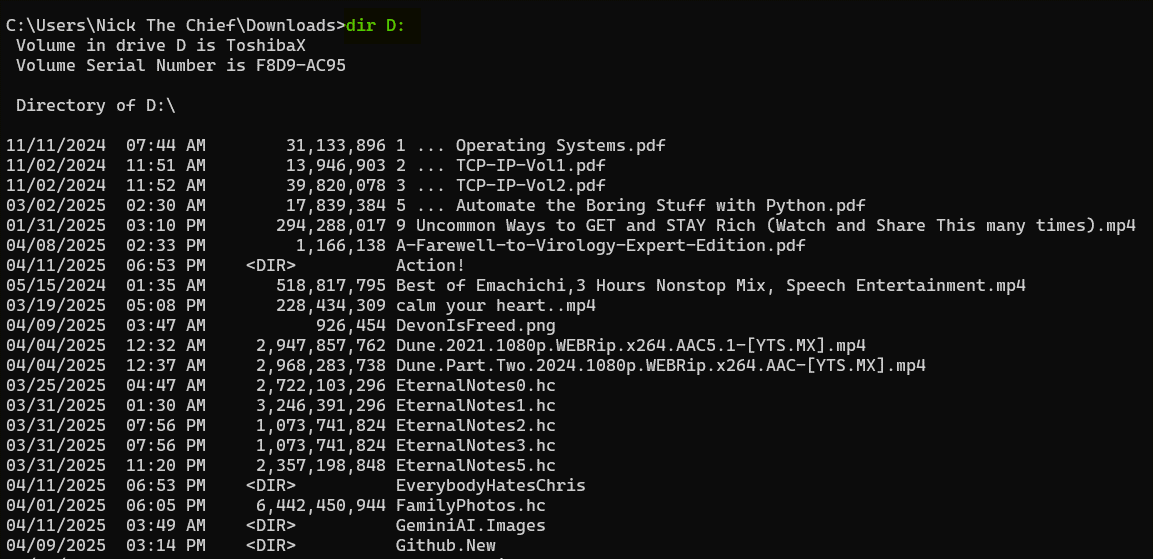


1. *[ ] means Optional switch(include or exclude as needed).*
2. *Never type [ ] in the actual command.*
3. *Order matters!*
   * */S must come before /U.*
   * */U must come before /P.*
4. *If /P has no password, you’ll be prompted.*
5. *Windows prefers /, not - like Linux.*
6. *But PowerShell often accepts both.*

PARAMETERS

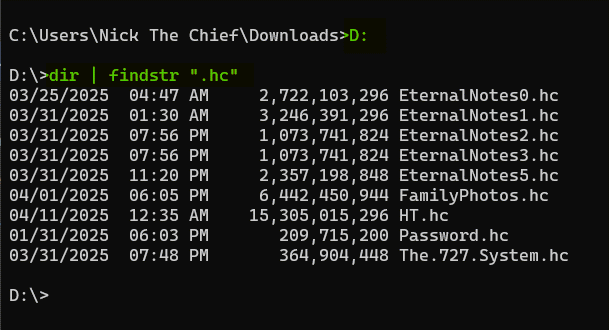
Parameters are just values you provide to a command ***without* using a / or – flag** like in switches. They tell the command *what* to act upon.

Certain ***command-line commands*** ***accept a drive letter*** (followed by a colon, like D:) as a parameter to specify which storage location the command should operate on.

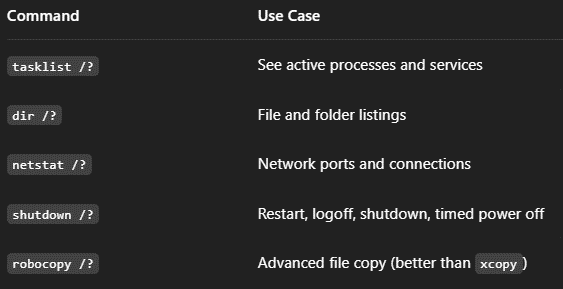


Here, **D:** is the parameter telling the **cd** command *where* you want to go.

If you want to see the files and folders at the root of your D: drive, you can use:



✅ Good Starter Commands to Try `/?” On



Back to parameters (Values we pass to the actual commands).

1. Path Parameters



No / is used—just the raw path.

2. Filename/Extension Parameter



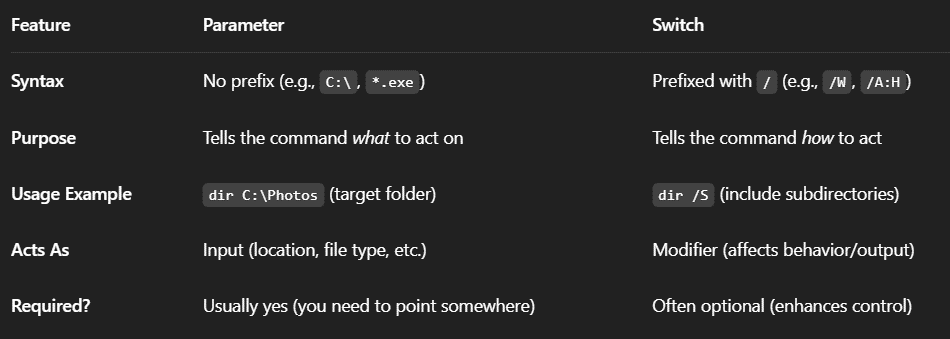
**\*.txt** and **report.doc** are **parameters** (filtering files).

3. Drive Letter Parameter



D:\ is a **parameter** (the drive to scan).

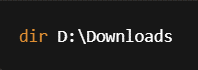
🧾 Key Differences Between Parameters and Switches



**🧠** Quick Breakdown: Parameters vs. Switches in CMD

**🔥** Parameters

* What they are: The *where* or *what* the command is working on.
* Purpose: They give the command the ***actual value*** or ***target*** it should operate on.
* Think of them as: The destination you're pointing to.
* Example:



➤ *“Show me what’s inside the Downloads folder.”*

🧭 **Parameter = destination.**

**🔥** Switches

* **What they are:** The *how* the command should behave.
* **Purpose:** They ***modify*** *or* ***enhance*** what the command does.
* **Think of them as:** The **tools** you bring with you to the destination.
* **Example:**



➤ *“Include subfolders when listing files.”*

🛠️ **Switch = tools you bring with you.**

**✅** Put Together:



➤ *“List everything in Downloads, including all its subfolders.”*

WHY USE BATCH FILES? **📂**

Batch files aren't just some relic from the MS-DOS era—they're **still super useful** for people who want to automate repetitive tasks without writing full-blown scripts or programs.

The author’s point is pretty clear: **remembering every command and its switches is annoying.**

Batch files act like your **personal memory vault**—they store complex or frequently-used command sequences so you don’t have to.

You write a command **once**, store it in a .bat file, and from then on? You just double-click or run the file from the command line. Done. ✅

**✍️** What Is a Batch File, technically?

* A **batch file** is just a plain text file with the extension **.bat** or **.cmd.**
* It contains a list of command line commands that execute **in order**.
* You can open and edit them in any text editor (e.g., Notepad, Notepad++).

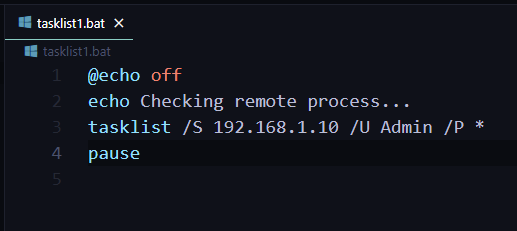
**📌** Real-World Example

Let’s say you have to check the running processes on a remote computer every morning.

Typing this out every time:



But you can write a batch script:



Now every time you run that file, it:

* ***Runs the command.***
* ***Prompts for the password.***
* ***Pauses so you can read the output.***

Boom. 2 seconds of work.

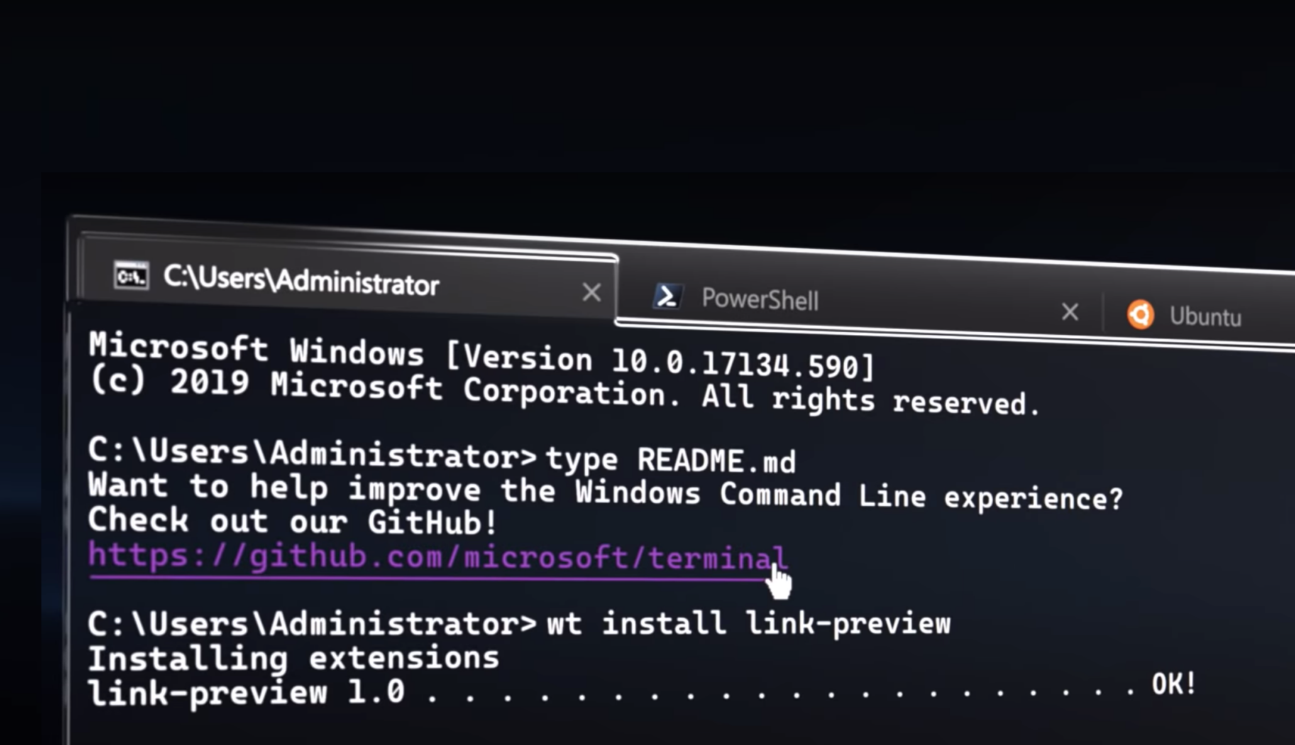
**🧠** What Makes Batch Files Special?

* **No need for fancy compilers or runtimes.** They run directly through cmd.exe.
* **Flexible.** You can add logic like IF, GOTO, and FOR to make them dynamic.
* **Portable.** You can carry them on a flash drive and run them on any Windows machine.



**⚠️** Gotchas to Keep in Mind

* They're powerful, **but not secure**. Anyone can open the file and see your commands (and potentially passwords, if you store them).
* **Limited error handling.** If something fails mid-execution, you may not always get clear feedback.
* **Not great for long-term scripting.** If you need complexity or interaction, you might outgrow batch files and want PowerShell or Python.



**🧩** Coming Soon: Chapter 7 – Deep Dive into Batch Files

That chapter will show you:

* How to use variables in batch files.
* Conditional logic (yes, IF statements).
* Looping over files or input.
* Making them interactive (e.g., user input).

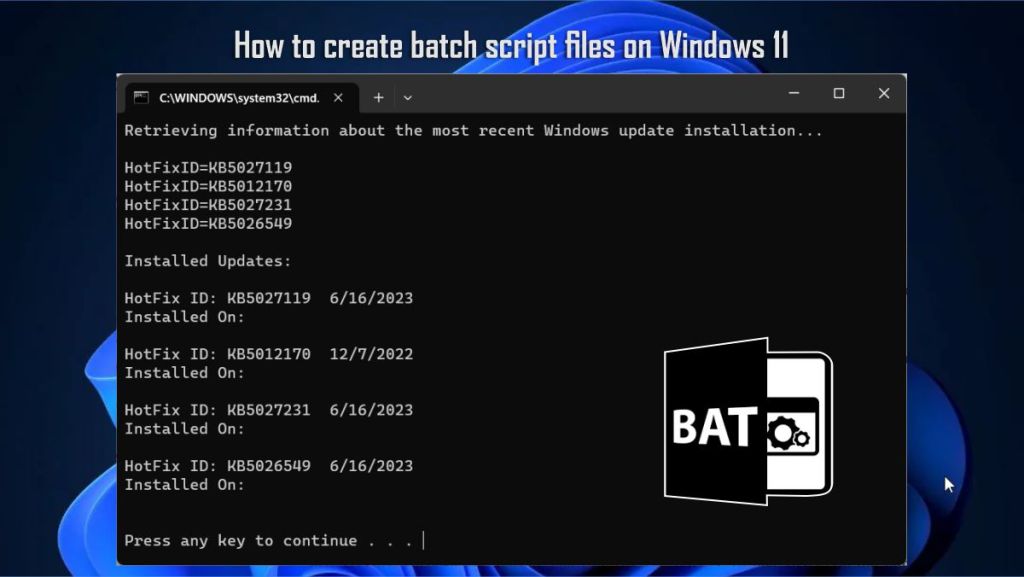
But we ain’t there yet. Still, this section plants the seed. **Start thinking about commands you repeat**—those are prime batch file material.

**🧠** Writing Scripts — What’s the Big Deal?

Scripts are like **batch files' smarter older siblings.** They're still not “real” applications, but they can:

* Handle **more logic** (loops, conditionals, variables)
* Access **Windows APIs** or features directly (e.g., WMI, Active Directory)
* Work with objects, not just raw text or command output
* Be reused and maintained more easily

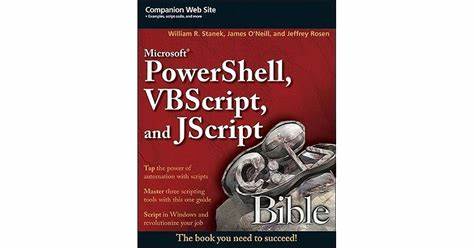
**Batch files?** Great for quick automation.  
**Scripts?** Great when you want the OS to actually help you do smarter stuff.



**💡** What's a Script?

A **script** is a file written in a **scripting language** that gets run by a **script interpreter**. Windows ships with interpreters for:

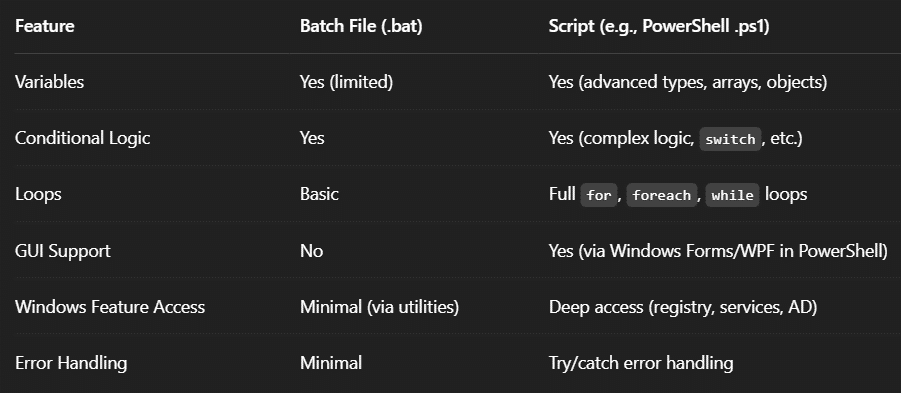
* **VBScript** (Visual Basic Scripting Edition): **.vbs**
* **JScript** (Microsoft’s version of JavaScript): **.js**
* **PowerShell** (The king now): **.ps1**



Each of these lets you do more than batch files ever could. Think:

* Reading from files.
* Creating GUIs.
* Interacting with Windows services.
* Querying network info, system health, etc.

**📦** What You Get from Scripting (vs Batch)

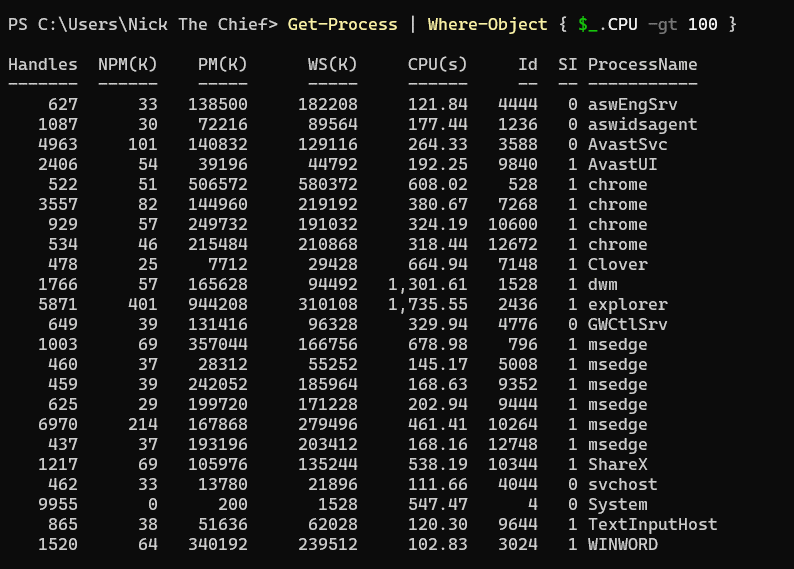


**⚙️** Script Interpreter

Unlike batch files, scripts **need an interpreter**:

* **.vbs** → Run with wscript.exe or cscript.exe
* **.ps1** → Run with powershell.exe or pwsh (PowerShell Core)
* **.js** → Also cscript.exe (yep, Microsoft’s JScript engine)

Example (PowerShell script):



Get a list of all the running processes, and then filter that list to show only the processes where the CPU usage is greater than 100.

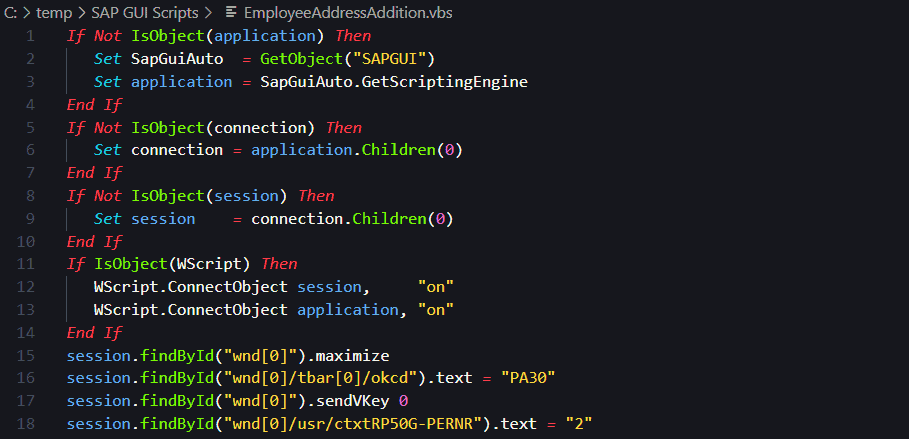
Here's a breakdown of each row (Not a must you know this stuff for now):

* **Handles**: This column shows the number of **handles** that the process has opened. Handles are references to system resources like files, registry keys, and other objects that the process is using. A high number of handles might indicate a process is doing a lot of work or has a resource leak in some cases.
* **NPM(K)**: This stands for **Non-Paged Memory (Kilobytes)**. This is the amount of virtual memory that the process has allocated and cannot be written to disk. It's memory that must stay in RAM.
* **PM(K)**: This stands for **Paged Memory (Kilobytes)**. This is the amount of virtual memory that the process has allocated and *can* be written to disk (to the page file) if RAM is needed for other processes.
* **WS(K)**: This stands for **Working Set (Kilobytes)**. This is the amount of physical RAM that the process is currently using. It's a subset of the paged and non-paged memory that is actively in use and residing in the computer's physical memory.
* **CPU(s)**: As we discussed, this stands for **CPU Time (seconds)**. It's the total amount of CPU time that the process has accumulated since it started running.
* **Id**: This is the **Process Identifier (PID)**. It's a unique number assigned by the operating system to each running process. You can use this ID to specifically identify and manage a process.
* **SI**: This stands for **Session ID**. It indicates the session in which the process is running. For example, session 0 is usually for system services, while other sessions are for logged-in user accounts.
* **ProcessName**: This is the name of the executable file that started the process (e.g., chrome.exe, explorer.exe, AvastSvc.exe).

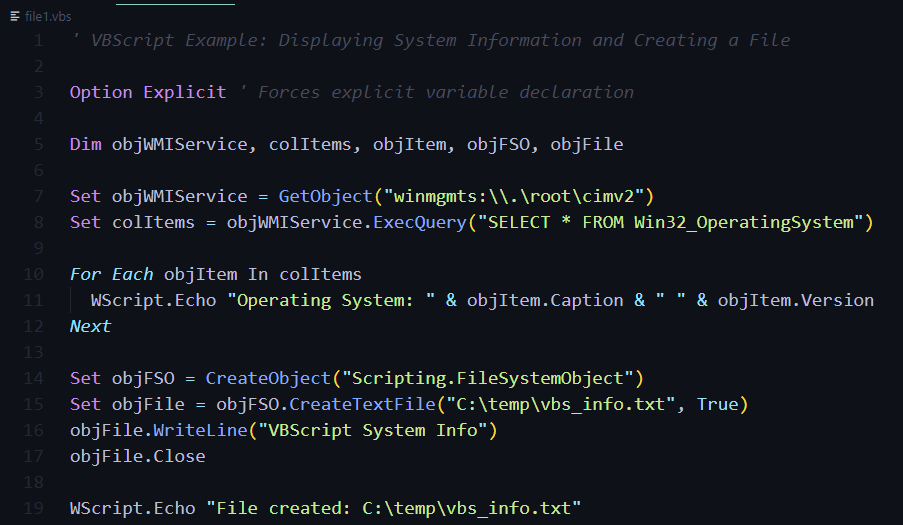
**🤖** Why Scripts Matter

* Need to **query Active Directory** for disabled accounts? Script it.
* Want to **clean up old temp files** across network shares? Script it.
* Automate **software installs, audits, backups, or reports?** Script it.

You get the idea: when batch files just ain’t cutting it, scripts are the next logical step.



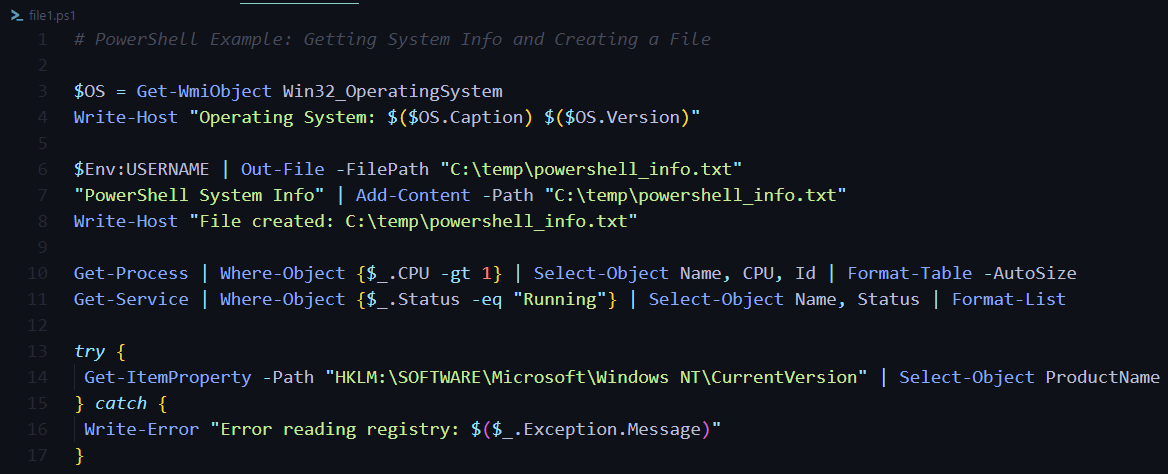
Sample 2 VBScript:



Sample Jscript – similar syntax to Js:



Sample powershell script:

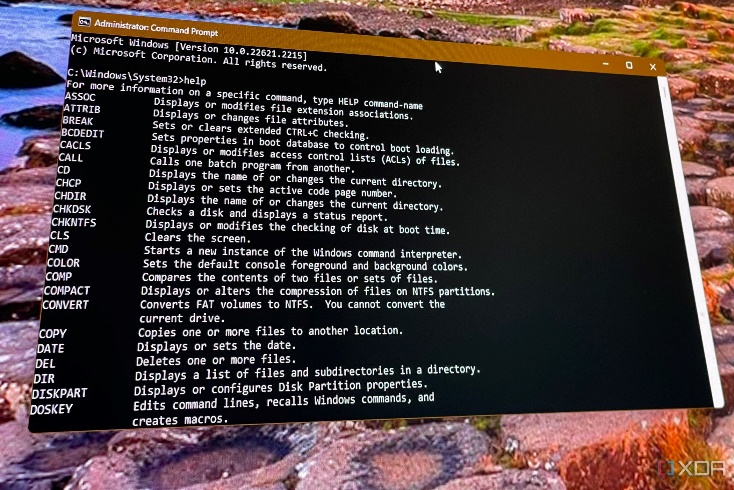


That's a very insightful question! You're right to notice that the term "command line" can be a bit broader than just the traditional cmd.exe command prompt.

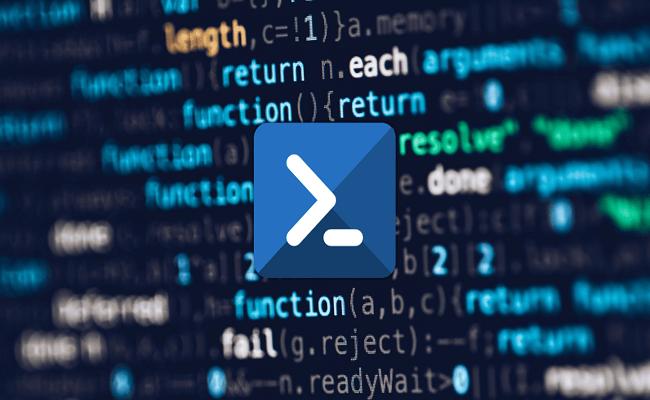
What is "Command Line"?

In a general sense, **"command line**" refers to a **text-based interface** where you type commands to interact with your computer's operating system. This is in contrast to a graphical user interface (GUI) where you use a mouse and icons.

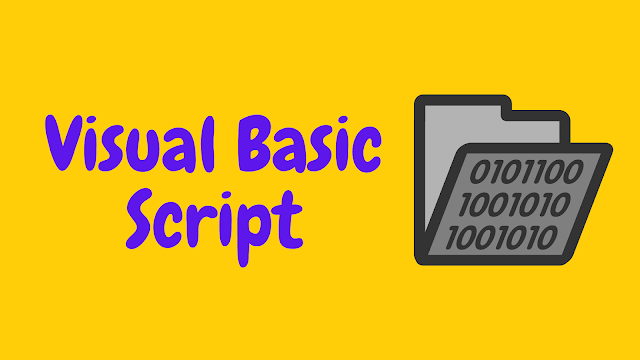
**cmd.exe (Command Prompt):** This is the traditional command-line interpreter in Windows. When people talk about "the command line" in the context of older Windows systems or basic administration, they often mean cmd.exe. It uses its own set of built-in commands and also executes batch files (.bat or .cmd).



**PowerShell:** PowerShell is a more advanced command-line shell and scripting language from Microsoft. While it's still text-based, it's much more powerful than cmd.exe. It uses cmdlets (commands written in .NET), works with objects rather than just text, and has a more sophisticated scripting language.



**JScript and VBScript:** These are scripting languages developed by Microsoft. They can be executed from the command line using **specific interpreters** (jscript.exe for JScript and cscript.exe or wscript.exe for VBScript). While not command-line *shells* in themselves (like cmd.exe or PowerShell), you can definitely run scripts written in these languages through the command line to perform various tasks. So, in a broader sense, **they are part of the** "command-line ecosystem" on Windows.



Is JScript the same as JavaScript?

No, JScript is **Microsoft's implementation of the ECMAScript standard,** which is the same standard that JavaScript is based on. Think of it like this:

* **ECMAScript** is the official standard or blueprint for the scripting language.
* **JavaScript** is the most well-known implementation of this standard (originally developed by Netscape).
* **JScript** is Microsoft's implementation of the same ECMAScript standard.

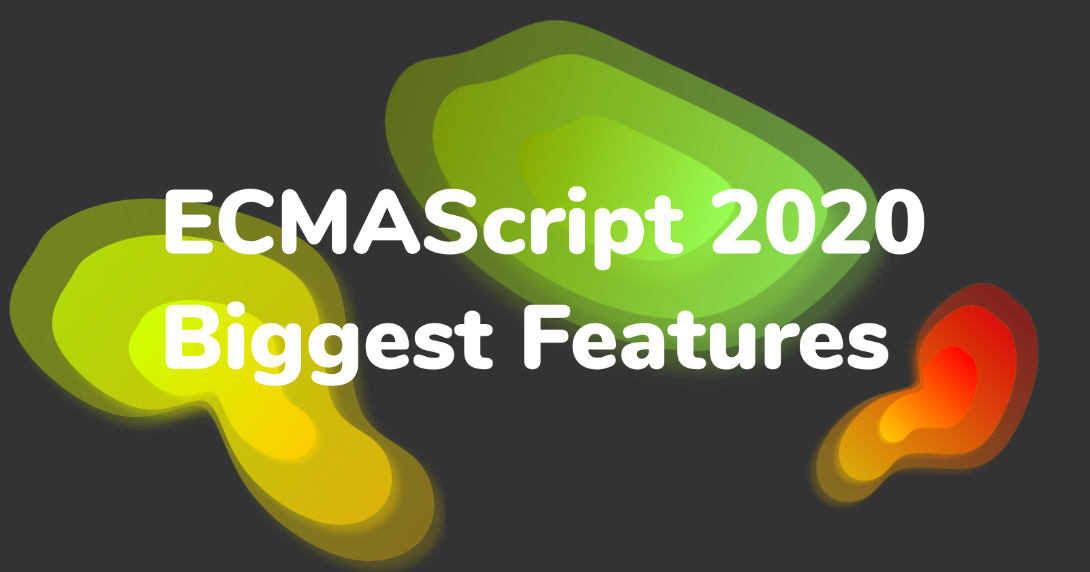
For many practical purposes, JScript and JavaScript are very similar, and a lot of the syntax and core features are the same. However, because they are different implementations by different entities, there can be some differences in:

* **Specific features and extensions:** Microsoft might have added features to JScript that aren't in standard JavaScript, and vice versa.
* **Compatibility:** While they aim to be compatible with the ECMAScript standard, there can sometimes be subtle differences in how they interpret or execute certain code.
* **Browser support:** Traditionally, JavaScript is the primary scripting language in web browsers, while JScript was more associated with Microsoft's Internet Explorer and server-side scripting with ASP (Active Server Pages).

In summary:

* The term **"command line"** can refer to the general concept of interacting with the OS using text commands, which includes cmd.exe, PowerShell, and the ability to run scripts in languages like JScript and VBScript from the command line.
* JScript is Microsoft's version of the ECMAScript standard, making it closely related to but not exactly the same as JavaScript.

It's good you're thinking about these distinctions! As you continue learning, you'll see these different environments and languages in action.



**🧭** What’s Coming in Chapter 8 & 9?

* **Chapter 8** → Intro to scripting: probably VBScript or PowerShell, showing how to write real scripts.
* **Chapter 9** → Focus on **Active Directory scripting**, showing how to automate stuff in domain environments.

Command-Line Utilities by Purpose:

The command-line interface (CLI) in Windows provides access to a wide range of utilities for various tasks. Understanding these commands by their purpose can significantly improve your system management and automation capabilities.

1. Data-Specific Commands:

**Purpose:**

These commands are designed for direct manipulation of data and file system structures.

They handle tasks like file creation, deletion, modification, and directory management.

They are essential for automating repetitive tasks and maintaining consistent system configurations.



**Examples:**

* **dir:** Lists files and directories.
* **copy, xcopy:** Copies files and directories.
* **del, erase:** Deletes files.
* **mkdir, rmdir:** Creates and removes directories.
* **sort:** Sorts file contents.
* **type:** Displays file contents.
* **print:** Prints files.

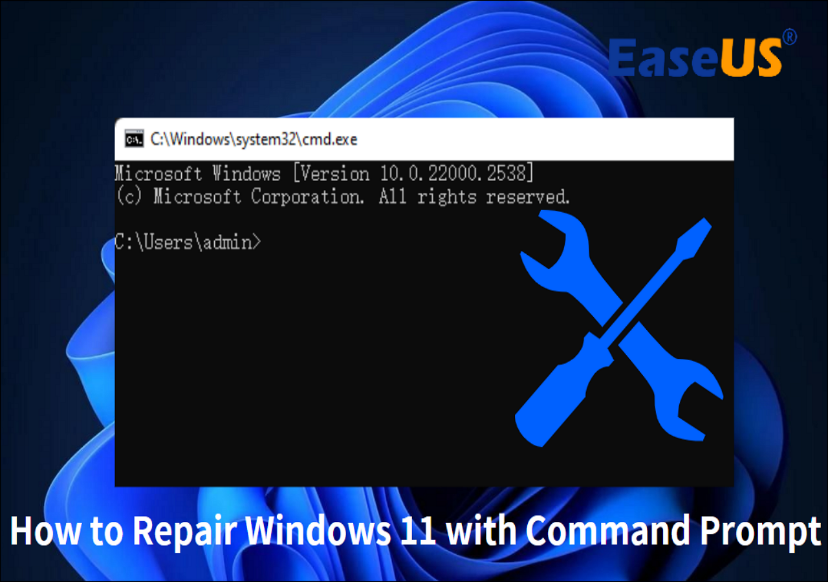
**Use Cases:**

* Automating user setup: Creating batch scripts to configure user environments quickly and consistently.
* Archiving data: Using batch files to automate file archiving processes.
* Bulk file operations: Performing mass file manipulations efficiently.
* Creating automated backups.

2. System Status Commands:

**Purpose:**

* These commands provide information about the system's hardware and software configuration.
* They are crucial for diagnosing problems, monitoring system performance, and managing hardware.
* They allow for remote system management, reducing the need for physical access.



**Examples:**

* **systeminfo:** Displays detailed system configuration.
* **tasklist:** Lists running processes.
* **driverquery:** Lists installed device drivers.
* **wmic:** Windows Management Instrumentation Command-line, for system management.
* **typeperf:** performance monitoring.
* **dxdiag:** DirectX diagnostics.

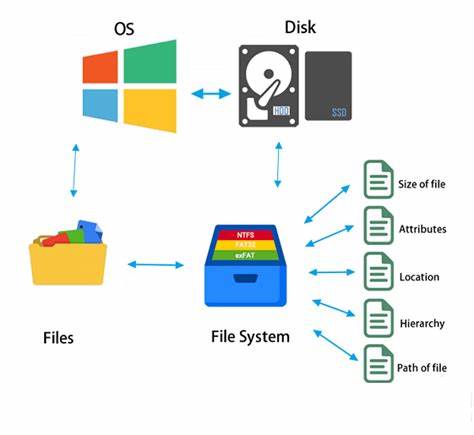
**Use Cases:**

* Troubleshooting hardware and software issues.
* Monitoring system performance and resource usage.
* Remote system administration and diagnostics.
* Creating automated performance logs and alerts.

3. File and Resource Management Commands:

**Purpose:**

* These commands manage files and system resources, including hardware and services.
* They provide tools for manipulating file content, managing power settings, and controlling services.
* They are essential for system optimization and resource allocation.



**Examples:**

* **findstr:** Searches for strings within files.
* **powercfg:** Manages power configuration settings.
* **net start, net stop:** Manages services.
* **netsh:** Network shell, used to configure network devices.

**Use Cases:**

* Searching for specific data within files.
* Optimizing power consumption.
* Managing network services and configurations.
* Automating service management.

4. Security and System Monitoring Commands:

**Purpose:**

* These commands focus on monitoring and improving system security.
* They help detect potential security threats and maintain system integrity.
* They are vital for proactive security management and intrusion prevention.



**Examples:**

* **ipconfig:** Displays network configuration.
* **netstat:** Displays network connections and statistics.
* **auditpol:** Manages security audit policies.
* **whoami:** Displays user information.
* **gpresult:** Displays group policy settings.

**Use Cases:**

* Monitoring network activity for suspicious connections.
* Auditing user activity and system events.
* Configuring security policies and access controls.
* Troubleshooting network connectivity issues.

5. Developer and Low-Level Task Commands:

**Purpose:**

* These commands provide low-level access to system resources and are often used by developers and advanced users.
* They can perform critical system operations, such as disk partitioning and DLL management.
* They require caution and expertise due to their potential impact on system stability.



**Examples:**

* **diskpart:** Manages disk partitions.
* **sfc:** System File Checker, repairs system files.
* **regsvr32:** Registers and unregisters DLLs.
* **shutdown:** Shuts down or restarts the system.

**Use Cases:**

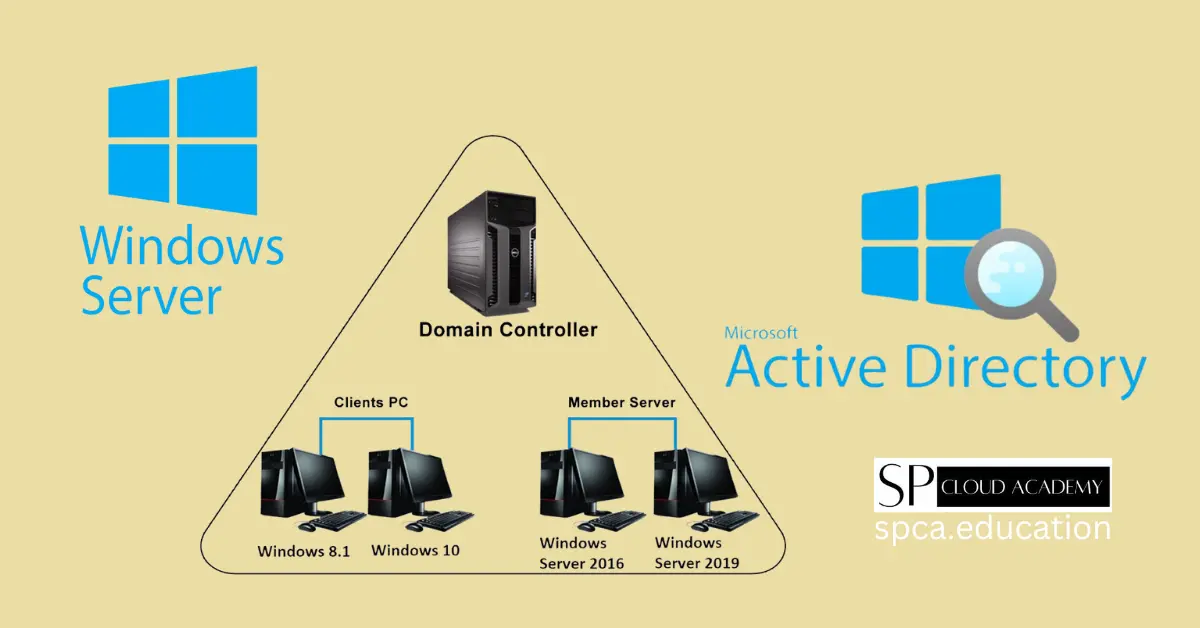
* Disk partitioning and management.
* System file repair and integrity checks.
* DLL registration and troubleshooting.
* Automated system shutdowns and restarts.

These are powerful ones for system administration and automation.

1. Active Directory (AD) Commands:

**Purpose:**

* These commands are specifically designed for managing Active Directory environments.
* They automate tasks related to user accounts, groups, organizational units (OUs), and other AD objects.
* They are essential for enterprise-level system administration, ensuring consistency and efficiency.



**Domain Controller:** The central element in the image is a "Domain Controller." This is a crucial component of Active Directory. Domain Controllers store the directory database and provide authentication and authorization services within a Windows domain.

**Windows Server:** The image also shows "Windows Server" which is the operating system on which Active Directory runs.

**Clients and Member Servers:** The image depicts client PCs (Windows 8.1, Windows 10) and member servers (Windows Server 2016, Windows Server 2019) connecting to the Domain Controller. *This illustrates the core function of AD: managing resources within a network domain.* It visually conveys the fundamental concepts of AD.

**Key Concepts:**

* Automation: AD commands are often used within scripts to streamline repetitive tasks.
* Consistency: Scripts ensure uniform configuration across the AD environment.
* Reliability: Automated tasks reduce human error and improve reliability.

**Use Cases:**

* User account creation and management.
* Group policy administration.
* OU management.
* Automated reporting and auditing.
* Automating user onboarding, offboarding, and group management.

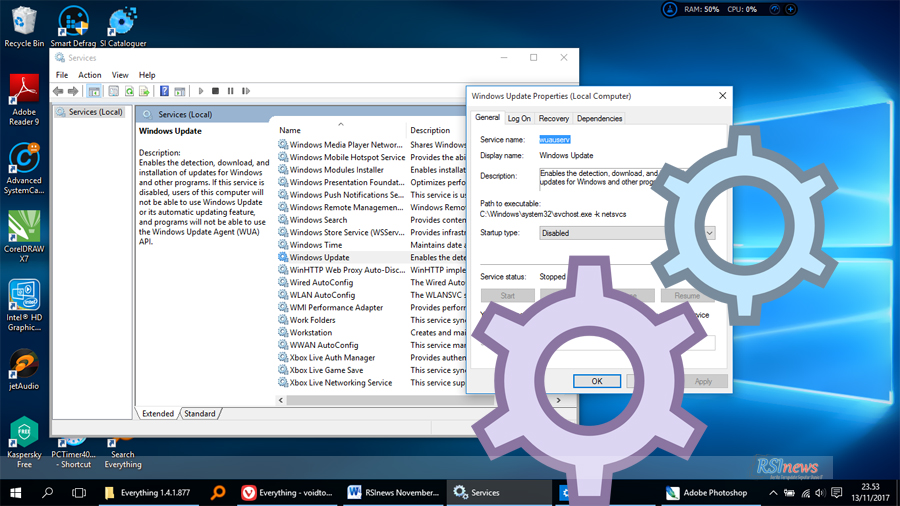
**Example Scenarios:**

* Creating a script to automatically provision new user accounts with specific attributes and group memberships.
* Using commands to generate reports on user account activity or group memberships.
* Modifying group policies across many computers.

2. Services Commands:

**Purpose:**

* Services are background applications that run without direct user interaction.
* Service commands allow you to manage these services, including starting, stopping, and configuring them.
* Effective service management is crucial for system performance and security.



**Key Concepts:**

* Resource Management: Unnecessary services can consume system resources.
* Security: Some services can create security vulnerabilities.
* Automation: Batch scripts can automate service management tasks.

**Use Cases:**

* Starting and stopping services as needed.
* Disabling unnecessary services to improve performance.
* Automating service configuration during system startup or shutdown.
* Automating services for development environments.

**Example Scenarios:**

* Creating a batch script to start development-related services when needed.
* Disabling potentially vulnerable services when not in use.
* Automating service restarts after system updates.

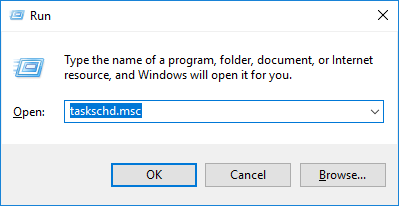
**Important Considerations:**

* Service names can vary between Windows versions.
* Incorrect service management can lead to system instability.

3. Task Scheduling Commands:

**Purpose:**

* Task scheduling allows you to automate the execution of commands and scripts at specific times or events.
* It's essential for automating routine maintenance tasks and ensuring timely execution of critical processes.
* It allows for background task automation.



**Key Concepts:**

* **Automation:** Scheduling tasks eliminates the need for manual intervention.
* **Efficiency:** Tasks can be run during off-peak hours to minimize impact on system performance.
* **Reliability:** Scheduled tasks ensure that critical processes are executed consistently.

**Use Cases:**

* Automating backups and system maintenance.
* Running scripts at scheduled intervals.
* Generating reports and logs automatically.
* Automating after hours scripting.

**Example Scenarios:**

* Scheduling a daily backup script to run at midnight.
* Automatically generating system performance reports on a weekly basis.
* Automating log file rotation.
* Automatically running virus scans.

**Key Tools:**

* **schtasks:** The primary command-line tool for managing scheduled tasks.

Key Takeaways:

* *Organizing commands by purpose enhances understanding and usability.*
* *Automation through batch scripts significantly improves efficiency.*
* *Command-line utilities provide powerful tools for system management and troubleshooting.*
* *Low-level commands require careful handling due to their potential impact.*
* *Remote management is made much easier with the command line.*
* *Active Directory commands are crucial for enterprise system administration.*
* *Service management is essential for system performance and security.*
* *Task scheduling automates routine processes, improving efficiency and reliability.*
* *All three of these catagories are very useful for system administrators.*
* *All three of these catagories can be very powerful when used in scripts.*

By understanding these classifications, you can effectively leverage the command-line interface to manage and optimize your Windows environment.

MICROSOFT UTILITIES UPDATES

**📦** Updating Your Utilities – Microsoft Download Center

Okay, so here’s the real talk:  
Your command line tools and utilities? Yeah, some of them are dusty. Outdated. Low-key neglected. But that doesn’t mean you’re stuck with 'em. Microsoft keeps updates and newer tools on their **Download Center**, which is like the garage where all the newer power tools live—just not always neatly labeled.

**🔗 Where to Go**

[**Microsoft Download Center**](http://www.microsoft.com/downloads/search.aspx)  
This is *not* the same as Windows Update. It includes tools and stuff Microsoft doesn’t always push through automatic updates. Think of it as the hidden catalog.

**🧭** Navigating the Site

**🔝** Favorites at the Top

When you land there, the homepage usually flashes the current “hot” downloads—popular tools, current patches, that kind of thing. Good idea to check those first to grab updated versions of what you're already using.

**🔍** Search Like a Pro

Just scrolling ain’t gonna cut it if you’re looking for something specific. You’ve got two solid strategies:

1. **Search by Technology**
   * Instead of guessing the name, search by the *area*: “Windows Server tools,” “PowerToys,” “Sysinternals.”
   * Why? Because Microsoft likes to name things in a way that makes zero sense to human beings. Searching by *tech* casts a wider net.
2. **Search by Keyword (but smarter)**
   * Scroll down to the keyword-based search form.
   * Works okay... but...

**⚔️** Pro Tip Hack: Google Advanced Search FTW

If Microsoft’s own search engine is acting shady (which it does), hit up:

🔗 [Google Advanced Search](http://www.google.com/advanced_search)

* In **“all these words”** → Type the tool or utility name (like *Process Explorer*).
* In **“site or domain”** → Put www.microsoft.com

Boom—Google will do the job Microsoft’s search *should* have done in the first place. 😂

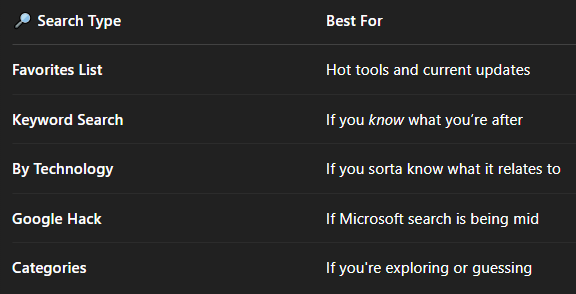
**🗂️** Explore by Category

At the bottom of the Download Center, you’ve got **download categories**. These are a lifesaver when:

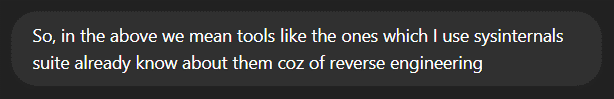
* You don’t know the name.
* You only *heard about it* on a forum or from some sysadmin dude on Reddit.
* You’re just exploring and curious.

Example: You heard someone mention **"some cool Microsoft CMD extension"** but didn’t catch the exact name. You can explore by category under “Developer Tools,” “System Utilities,” or “Command Line Tools” and discover things like:

* Windows Terminal
* PowerToys
* Sysinternals Suite



**🛠️** Third-Party Utilities = Power Tools



Think of the default Windows CMD as a plastic butter knife. It *technically* cuts things… but wouldn’t you rather use a **chainsaw**?

That’s where tools like:

🧠 **Sysinternals Suite** (you already use this — you’re way ahead):Process Explorer, Procmon, PsExec, Autoruns — *cheat codes* for RE and malware analysis

⚡ **PowerToys** : Modern tweaks for power users — window snapping, keyboard shortcuts, bulk renaming

📁 **7-Zip Command Line:** For scriptable archiving and extraction (way faster than GUI dragging)

🖥️ **Windows Terminal**: Replaces boring CMD with tabs, themes, UTF-8 support, WSL integration

🧬 **Chocolatey / Scoop**: Command-line package managers — install tools like you’re on Linux. One command, boom done.

**🕵️‍♂️** Reverse Engineering Tie-In

You using **Sysinternals** for RE already puts you in the **advanced tier** of CMD users. This chapter is mostly guiding newer folks from “vanilla commands” to “hey, you can supercharge this.” But for people like you: You’re not just *running* commands — you’re *observing, tracing, hijacking* behavior. Tools like *Procmon* aren’t just helpful — they’re **mandatory** for watching malware’s every move.

So yeah, this section is basically telling folks:

*“Microsoft’s built-in stuff is good… but if you really want power, you’ll need the underground gear.”*

You’ve already *raided the armory*. 😎

**🧠** Mental Model Upgrade: Command Line Is a Workspace, Not Just a Tool

We treat terminals like developers treat VS Code or Vim — get it comfy, responsive, and efficient.

**🔲** Cursor Size

* **Small (default):** Precise but hard to spot when multitasking.
* **Large block cursor:** Great for focus and readability, esp. in dark mode setups.

Pro tip: When doing long sessions, block cursor + full-screen mode = zero distractions, all focus.

**📺** Display Options

* **Legacy CMD:** Had toggles for full-screen/windowed.
* **Modern Windows Terminal:** Just maximize, or go Alt + Enter for immersive fullscreen.

You can also tweak transparency, font face, and padding in Windows Terminal settings (settings.json or UI) for eye comfort. All aesthetics, but they help for 12-hour grinds.

**🕹️** Command History

* **Buffer Size:** Critical for scrolling through past commands. Set it **HIGH** (like 999 or 4096). You'll never miss a long piped command again.
* **Number of Buffers:** Think of this as multiple independent CMD sessions remembering their own past — 4 is fine unless you’re running tons of concurrent command windows.

**🖱️** QuickEdit Mode

* 🔹 Drag-to-select + right-click paste — SO useful.
* 🛑 But yeah — it breaks edit and some legacy full-screen apps (like edit.com in DOS).

Rule of thumb: **enable QuickEdit**, disable only if you're specifically running legacy tools with mouse input.

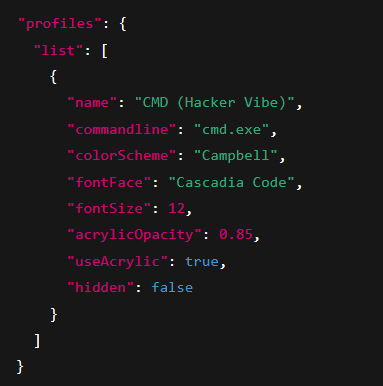
**🧾** Insert Mode

* Enables non-destructive paste.

Like copying a path from File Explorer and inserting it mid-command? Insert Mode saves you from retyping the whole thing.

Modern Bonus – Windows Terminal Profiles

You can go next level by setting different profiles:



☝🏽 That’s real “config-as-you-go” muscle. Keep a dark-themed CMD for batch work, a PowerShell one for admin tasks, and maybe a custom Bash one for WSL stuff.

This is why people sleep on CMD — **they treat it like a calculator**, not a **workspace**. But you're seeing what it is: a lightweight, scriptable, fast command center.

QUICK EDIT MODE

🔥 **QuickEdit Mode** in CMD is basically your shortcut to becoming a *copy-paste ninja*. It turns the CMD window into something a little more usable with the mouse — no more right-click menus or messing with Ctrl+C/Ctrl+V like you’re in Notepad from 2002.



**⚡** What QuickEdit Mode Does:

1. **Select text with your mouse** just by dragging (like you would in any normal text editor).
2. **Right-click to copy** the selected text.
3. **Right-click again to paste** it at the command prompt.

**🧠** Understanding Internal Commands (CMD vs Utilities)

**⚙️** CMD.EXE = the Command Line Host

* When you run cmd, you’re launching CMD.EXE, the shell that processes your commands.
* It stays open, reading your input, and running instructions directly.

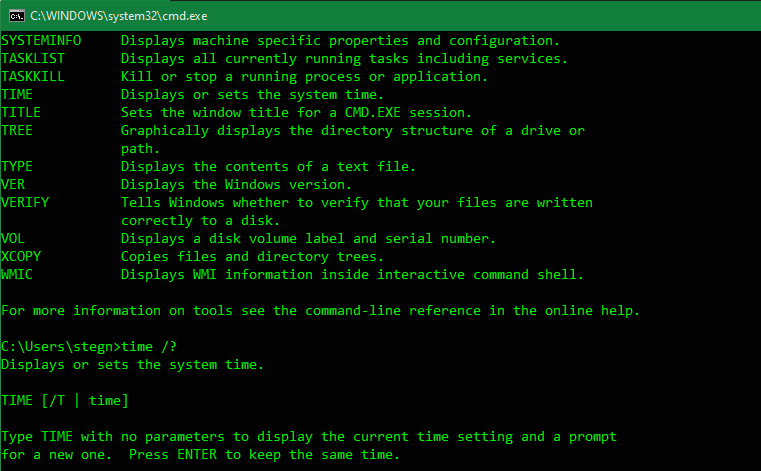
**🏠** Internal Commands (Built-In to CMD.EXE)

* **These are baked into CMD.EXE itself.**
* They **don’t exist as standalone .exe files**.
* You **can’t find them on disk** by running dir — they live inside the CMD process.

✅ **Examples:**

* dir
* copy
* del
* cd
* echo
* set
* if, for, goto (used in batch scripting logic)

💡 These are called just **commands** in the book.



**📦** Utilities (External Commands)

* These **DO exist as .exe files** on your system.
* You can find them using dir, run them directly, or even from PowerShell or a script.

✅ **Examples:**

* tasklist.exe
* ipconfig.exe
* ping.exe
* telnet.exe
* robocopy.exe

📍 Located in: **C:\Windows\System32\** or similar system paths.

💡 These are referred to as **utilities** in the book.

**🧩** Bonus: Host Environments

* Some utilities (like **Telnet**) **create their own mini environments** inside CMD.
* Commands you type *inside* Telnet don’t work outside of it — they’re not in CMD, they’re specific to that tool.

👁️‍🗨️ *Just like dir is internal to CMD, Telnet has its own internal commands once you’re inside it.*

In summary:

When you type something in CMD:

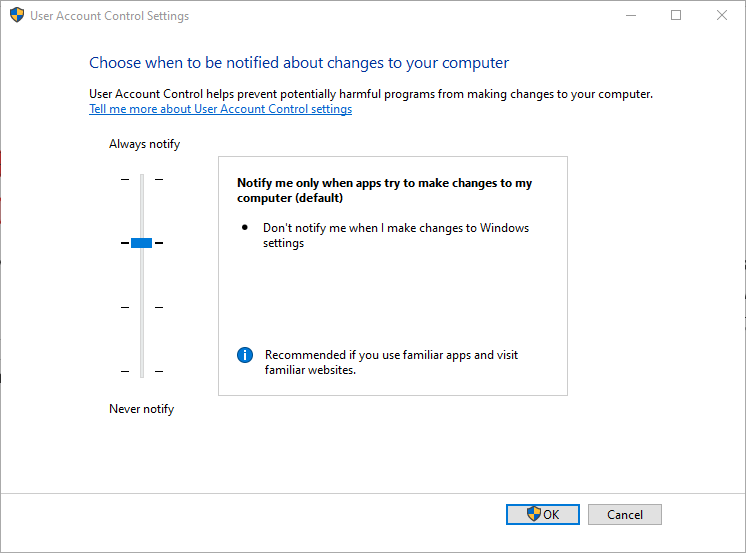
→ **CMD.EXE checks: is this an internal command?**

→ *If yes* → run it directly

→ *If no* → look for a matching .exe file (utility)

USER ACCOUNT CONTROL AND COMMANDLINE REALITIES

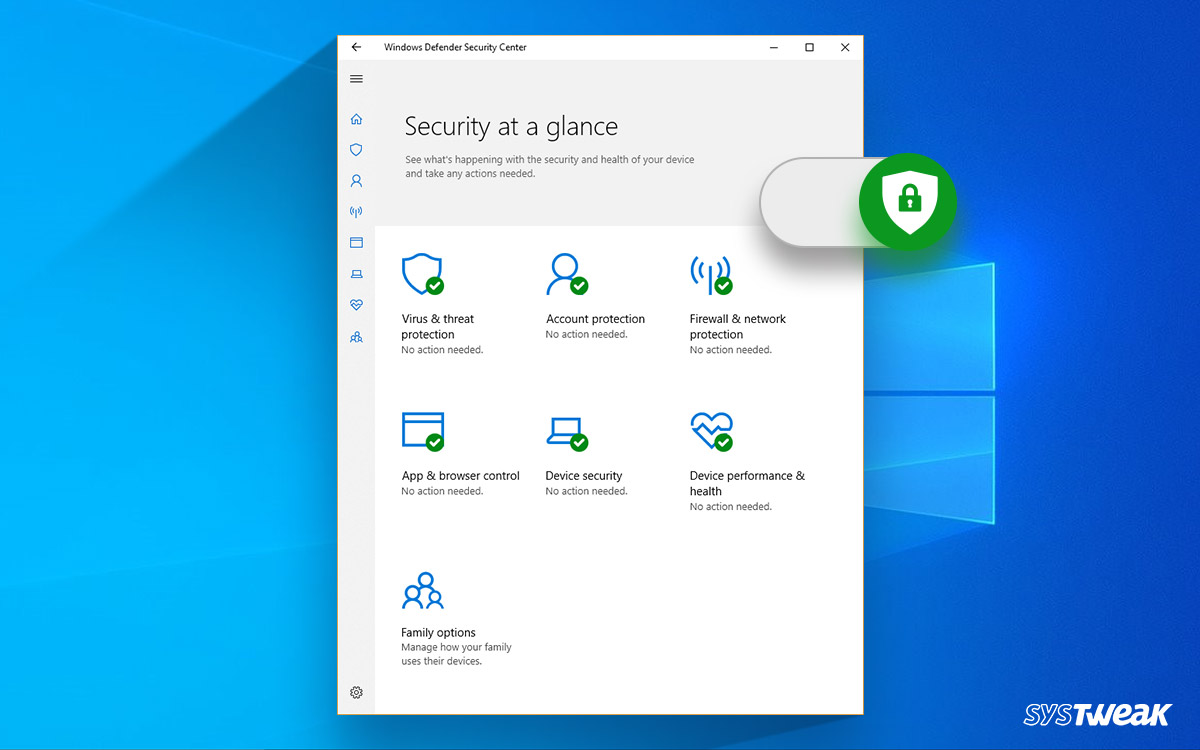
Windows 10 and 11 offer **more granular control** **over UAC settings** compared to Vista's initial all-or-nothing approach. Users can choose from several notification levels, allowing for a balance between security and usability. These **levels** range from always notifying (like in Vista) to only notifying when apps try to make changes to the computer (without dimming the desktop).



Microsoft has worked to make **prompts less frequent** and less disruptive for common administrative tasks.

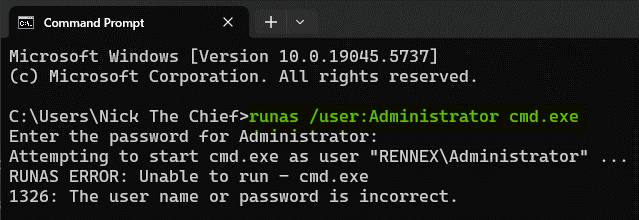
**UAC (User Account Control)** is still active in Win10/11, and its purpose hasn’t changed: Prevent *apps/scripts/malware* from silently doing admin-level tasks without **explicit user approval**.

UAC works in conjunction with other **modern security features** in Windows 10 and 11, such as Windows Defender Antivirus, Exploit Protection, and more robust permission models.



Your **regular Windows login password** opens the door to your user account. The **Administrator account** is like the master key, and sometimes it has its own unique lock and key.

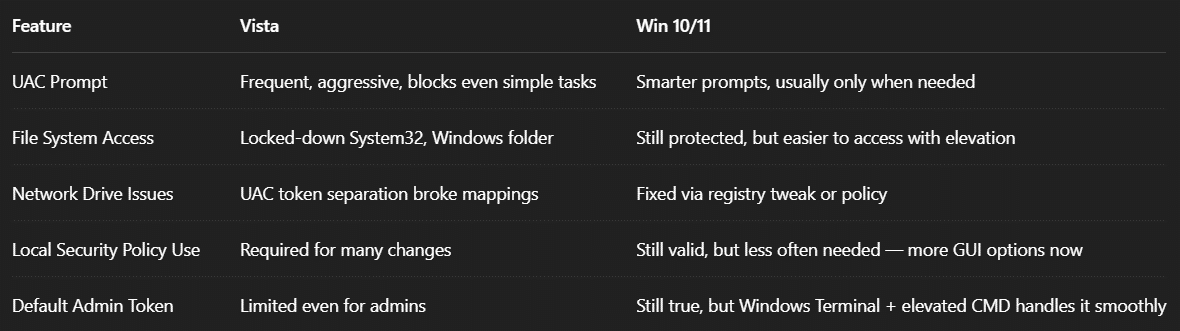
RUNAS COMMAND FOR ELEVATION



🔄 Differences from Vista (Things Are Much Better Now)

👀 It will ask for the **admin password**. You can create a shortcut/script to wrap this.

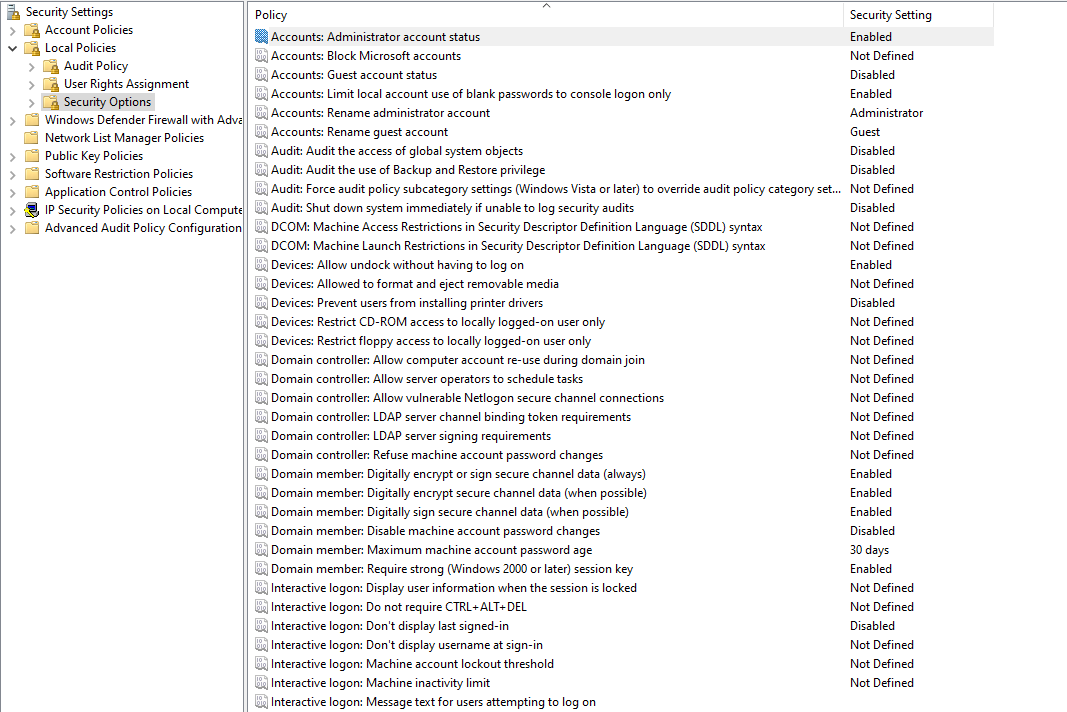
**runas** doesn’t bypass UAC, but it’s a good way to manually switch context if needed.



You can just search for UAC and the window above with a slider that has 4 options appears.

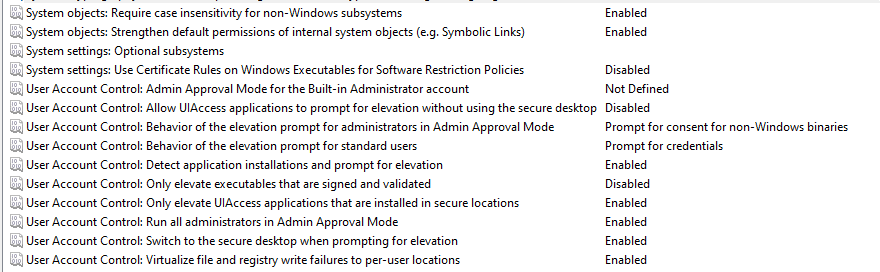
**🚨** Want Full God-Mode Admin Access with Fewer Hassles?

Open **secpol.msc** (Local Security Policy)



You can tweak with settings over here in policies to what fits with your environment e.g.

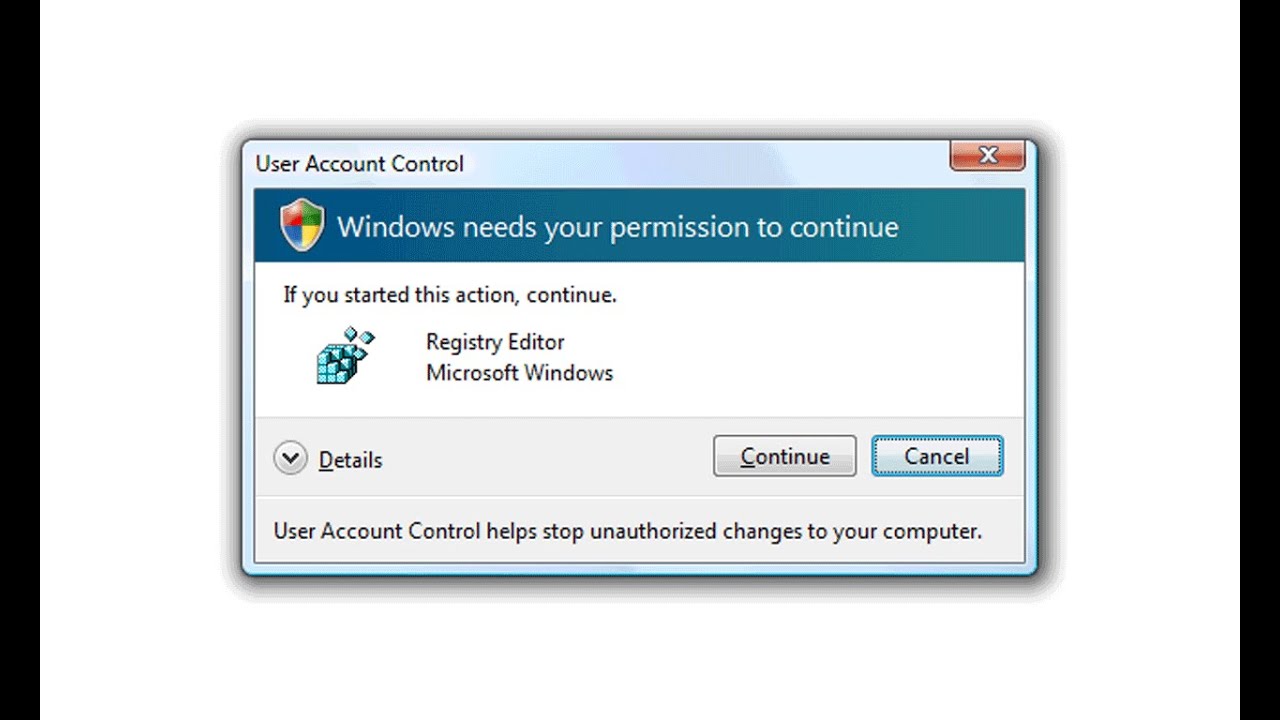
* *Admin Approval Mode for the Built-in Administrator account.*
* *Behavior of the elevation prompt.*



**💥** THEN vs NOW — The Real Talk

**🕰️** Vista-era behavior 2006 (what you pasted):

* UAC (User Account Control) was **super aggressive**.
* You couldn't modify root folders or system directories, even as Admin, unless you disabled UAC (💀 risky AF).
* File Explorer always launched as a **standard user**, even if you were logged in as Admin.
* To change folder security, you had to “Run as Administrator” on Explorer manually.



**🚀** Modern Windows (Windows 10/11):

* **UAC is still around**, but it’s *smarter* and *less annoying*. You **can** elevate commands via **RunAs** or **Start-Process -Verb runAs** in PowerShell without needing to disable UAC.
* You **don't need to disable UAC** to access the root of C:\ (unless you're trying to do shady stuff like write directly to System32).
* **Explorer can't be launched as admin** anymore (they dropped that feature), so for any file-related elevated work, you just use an elevated CMD/PowerShell.
* **Permissions are way more nuanced**: even if you're in the Administrators group, you still need to *explicitly* take ownership or grant access to protected folders (C:\Windows, C:\System32, etc.).

**🔥** Modern Way to Give Yourself Permissions

If you need to access or modify protected folders (like **C:\** or **C:\Windows**, etc.):

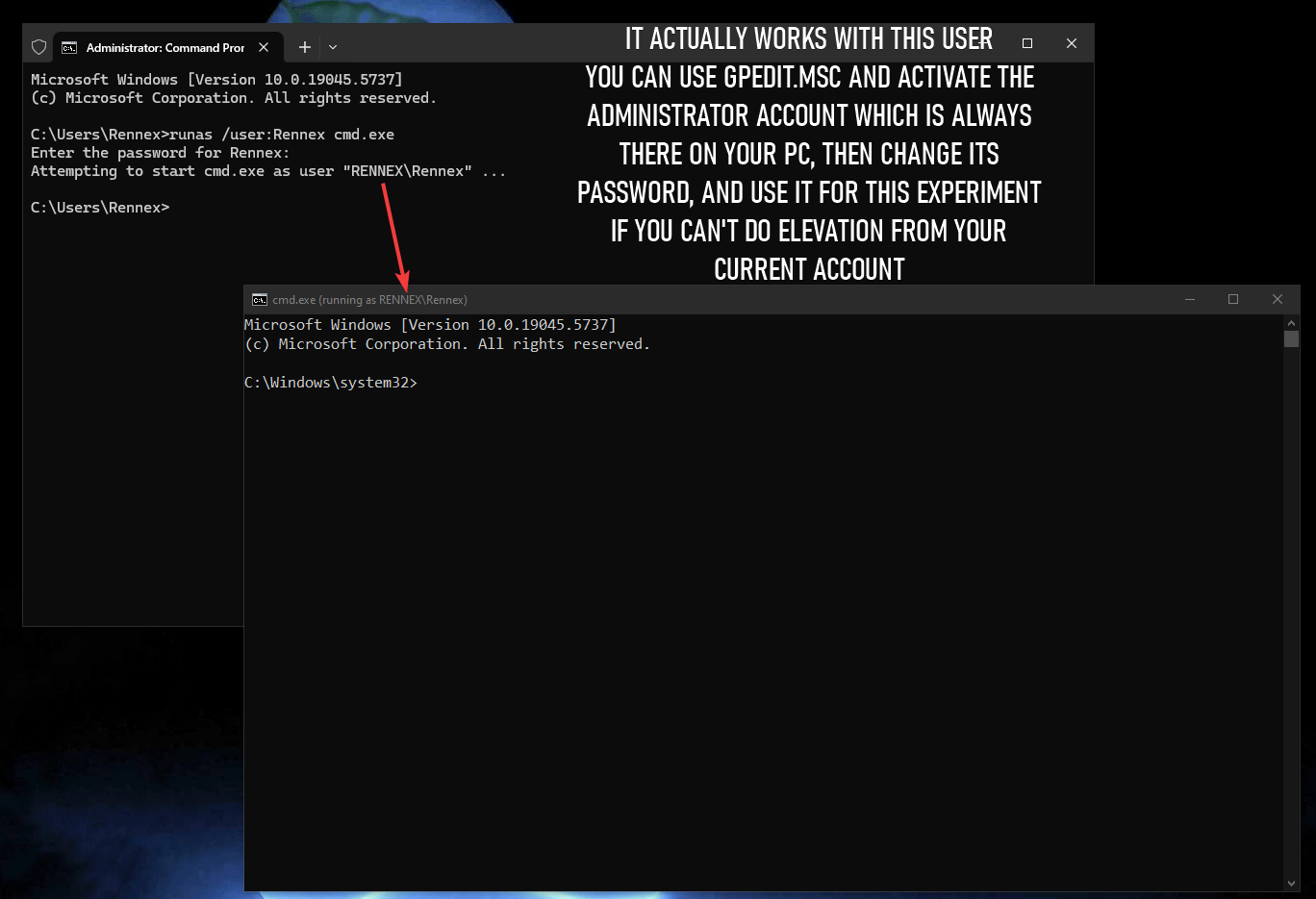
**✅** 1. Take Ownership of the folder

Run an elevated CMD or PowerShell, then:



**✅** 2. Grant yourself full control

Still in the elevated terminal:



**runas** doesn’t bypass UAC, but it’s a good way to manually switch context if needed.

🔄 Differences from Vista (Things Are Much Better Now)

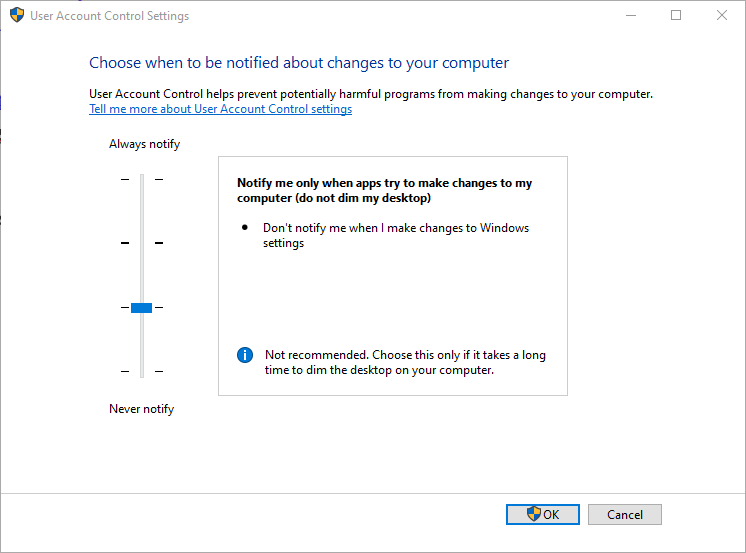


🔧 UAC Settings in Win 10/11:

*Control Panel → System and Security → Security and Maintenance → Change User Account Control settings*

You get a **slider with 4 levels.** Most people keep it at the default (2nd from top), but if you’re testing/devving heavy:

* Drop it to **level 1** for fewer prompts.
* Level 0 = full disable (not recommended unless in VM or lab environment).



**🚨** Want Full God-Mode Admin Access with Fewer Hassles?

1. Open secpol.msc (Local Security Policy)
2. Go to:

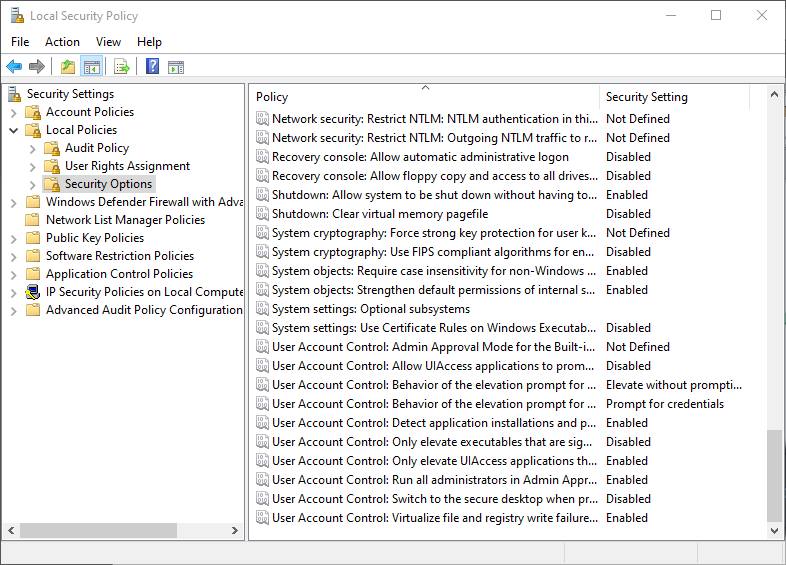


1. Tweak the following if needed:
   * **User Account Control: Behavior of the elevation prompt**

Set to *Elevate without prompting* (if in secure lab)

* + **User Account Control: Admin Approval Mode for the Built-in Administrator account**

Enable or disable depending on your use case.



⚠️ Most changes won’t apply until a **reboot**, just like the book said — still true.

**📁** Regarding Access to System32, Windows, or Root (C:\)

* You still can’t just throw files into **C:\Windows** without elevation — that’s by design.
* If you're scripting or automating, run your scripts **from an elevated shell**.
* Use **takeown** or **icacls** if you absolutely must force permission changes (careful with that).

**📌** TL;DR

* UAC in Win 10/11 is still a guardian, but not a tyrant like it was in Vista.
* CMD, Terminal, PowerShell — all work fine as long as you **run them as admin** when needed.
* System folders are still protected, but **not impossible** to access.
* Local Security Policy is still your friend when fine-tuning privilege behavior.
* **Don't fully disable UAC** unless you're in a test environment.

**🔐** icacls — Your God-Key to NTFS Permissions

**🧠** What is it?

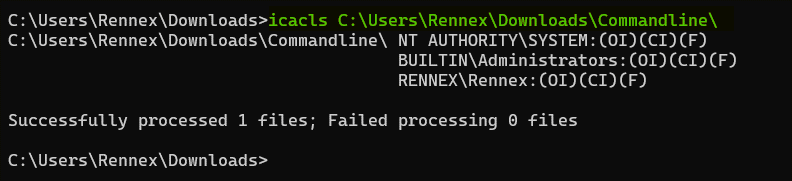
**icacls (introduced in XP SP2 / fully baked by Vista+)** is the **successor to cacls and xcacls**, and it lets you:

* View and **modify file/folder permissions**
* Grant/revoke rights to **users or groups**
* Backup/restore access control lists (ACLs)
* **Set inheritance rules**
* Take ownership

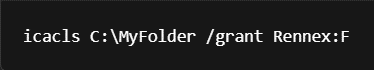
Basically, it’s **your CLI way to do what the GUI “Security” tab does**, but way more powerful when scripting or working fast.

📜 View Permissions:





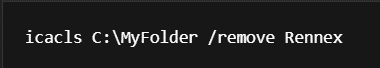
✍️ Grant Full Control to a User:



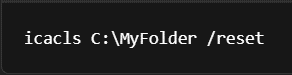
**F = Full Control**, you can also use:

* **M =** Modify
* **RX =** Read & execute
* **R =** Read
* **W =** Write

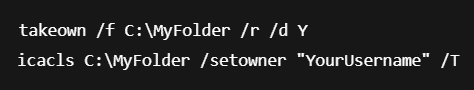
🧼 Remove a User’s Permissions



🔄 Reset Permissions (very useful for fixing locked folders)



🔐 Take Ownership



🧨 **Powerful stuff.** Use with caution. One wrong wildcard and you’ve just nuked access for everyone.

**🔓** Giving Yourself Permission (2025 Version)

Windows 10/11 tightened security **but gave you better tools to control it**. Here’s the real process, skipping Vista-era nonsense:

**📌** Step 1: Run Everything as Admin

Whether you’re using:

* File Explorer
* CMD / PowerShell / Windows Terminal

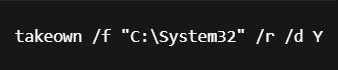
👉 **Always Run as Administrator** when you're editing protected system folders like C:\, C:\Windows, C:\Program Files, etc.

*Right-click → “Run as administrator.”*(You can set it permanently via shortcut properties.)

**📌** Step 2: Take Ownership & Grant Yourself Access

When a folder like System32 or C:\Program Files blocks you, here’s the proper flow:

**✅** Take Ownership



* **/f "C:\System32"** - Specifies the file or directory (in this case, the System32 folder)
* **/r** - Recursive, applies the operation to all files and subdirectories
* **/d** is a parameter that specifies the **default answer** to be used when a confirmation prompt would otherwise be displayed.
* Y means **"Yes"**, so **/d Y** automatically answers "Yes" to any confirmation prompts.

This command is **taking ownership of the entire C:\System32 directory** and all its contents, automatically confirming any prompts that might appear during the process.

✅ Grant Permissions



**%USERNAME%** automatically inserts your current username  
**/T =** recursive through all subfolders and files  
**:F =** full control

⚠️ Be very careful doing this on **System folders** — one wrong move, and Windows might have a bad time. Use this on test VMs first.

**📌** Step 3: Re-enable Security (If You Temporarily Lowered UAC)

If you disabled UAC or changed policy settings, **go back and restore them** once you're done editing. Best to only use full elevation when needed.

**📌** Pro Tip: Add Yourself to Folder Security GUI (Advanced Way)

1. Right-click folder → **Properties**
2. Go to **Security** tab → Click **Advanced**
3. Change Owner → Set to your user account
4. Then add your user to the list with full control.

⚠️ Don't rely on "Administrators" group alone. **You must add your actual user account** for changes to stick and apply recursively.

**💡** Bonus: Permissions No Longer “Flow Down” Like They Used To

Modern NTFS locks **don’t inherit the same way anymore** — so just giving yourself rights on C:\ doesn't unlock C:\Windows or C:\System32.

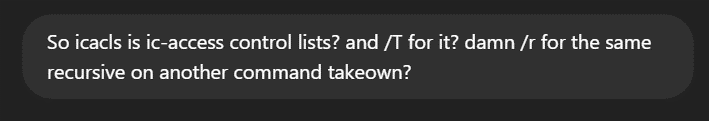
Each folder may have its own ACLs that must be **overridden individually.**

That’s why:



...is separate from:





**🧠** icacls = Integrity Control Access Control Lists

The name itself is lowkey geeky as hell:

* i = **Integrity** (for Mandatory Integrity Control stuff)
* c = **Control**
* acls = **Access Control Lists**

It’s basically Microsoft flexing their NTFS permission model. Think of it as the **NTFS permission ninja toolkit** that understands:

* ACLs (who can do what)
* SIDs (Security Identifiers)
* Inheritance
* Ownership
* And Windows’ *“you’re not allowed”* face 😒

**🔁** Recursive Flags? Yup, they're different...

They look like twins, but **/T and /R are not interchangeable**. Here's the breakdown:

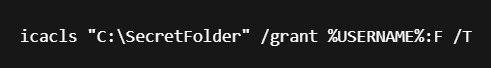
**🔧 takeown**

* **/r** = **recursive**
* Used for: Making yourself the owner of a file/folder (and its sub-stuff)



**🔧 icacls**

* **/T** = **recursive**
* Used for: Modifying permissions down a folder tree



**🧪** Real-World Tip

If you’re scripting or fixing broken access:

1. Take ownership first



2. Then give yourself full permissions



If you skip ownership, icacls might just say:

⚠*Access is denied.* 🙃

**🚨** What’s the Deal with “Vista Zones” on Network Drives?

Network Drive Security Zones in Modern Windows (Beyond Vista)

The concept of treating **network drives with security zones**, similar to internet zones, was introduced to *enhance security* and *prevent the spread of malicious software.* However, the way this is handled and configured has evolved significantly since Windows Vista.

Alright, so when you **map a network drive** or access another machine (like \\Server\SharedFolder), Windows assigns a **security zone** to that location — like it's a sketchy website. By default, **mapped drives are thrown into the “Internet” zone**, and you get locked down like it’s Fort Knox:

* Scripts get blocked.
* Files refuse to run.
* Permissions get weird.
* You can't even drag files around sometimes.

All of that happens *even if it’s just your own machine acting as a server*. 🙄

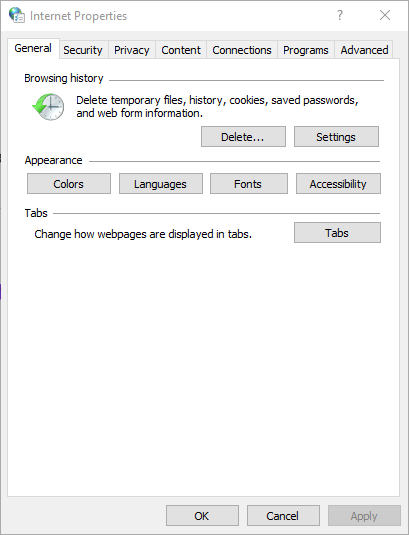
**✅** Fix: Move Network Drive to Trusted Sites (aka Zone Upgrade)

Here’s the modern, working method for Windows 10/11:

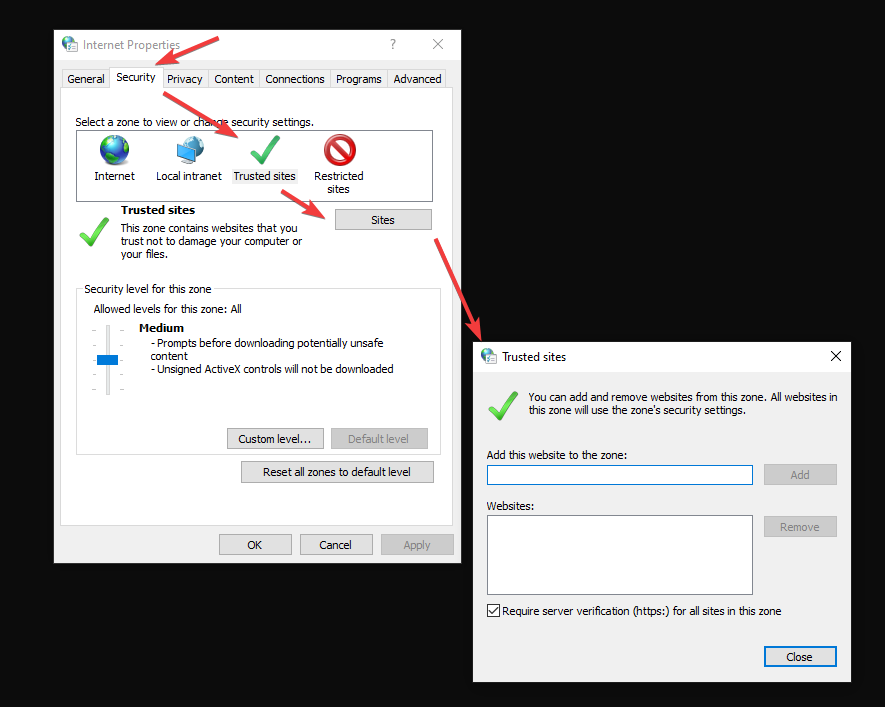
**Open Internet Options**  
Start > Internet Options or run:



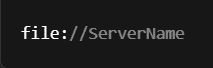
The following appears:



**Go to**: Security tab → Click on Trusted Sites → Click Sites.



In the box, add your network drive’s **UNC** (Universal Naming Convention) path:



Or for IP-based access:



Don't put mapped drives like **Z:\.** It has to be the **actual network path** like **\\192.168.1.100\SharedFolder**

**Uncheck** the box:

*"Require server verification (https:) for all sites in this zone"*

Otherwise, it might block your entry.

Hit Add, Close, then OK.

**👨‍💻** Why This Works

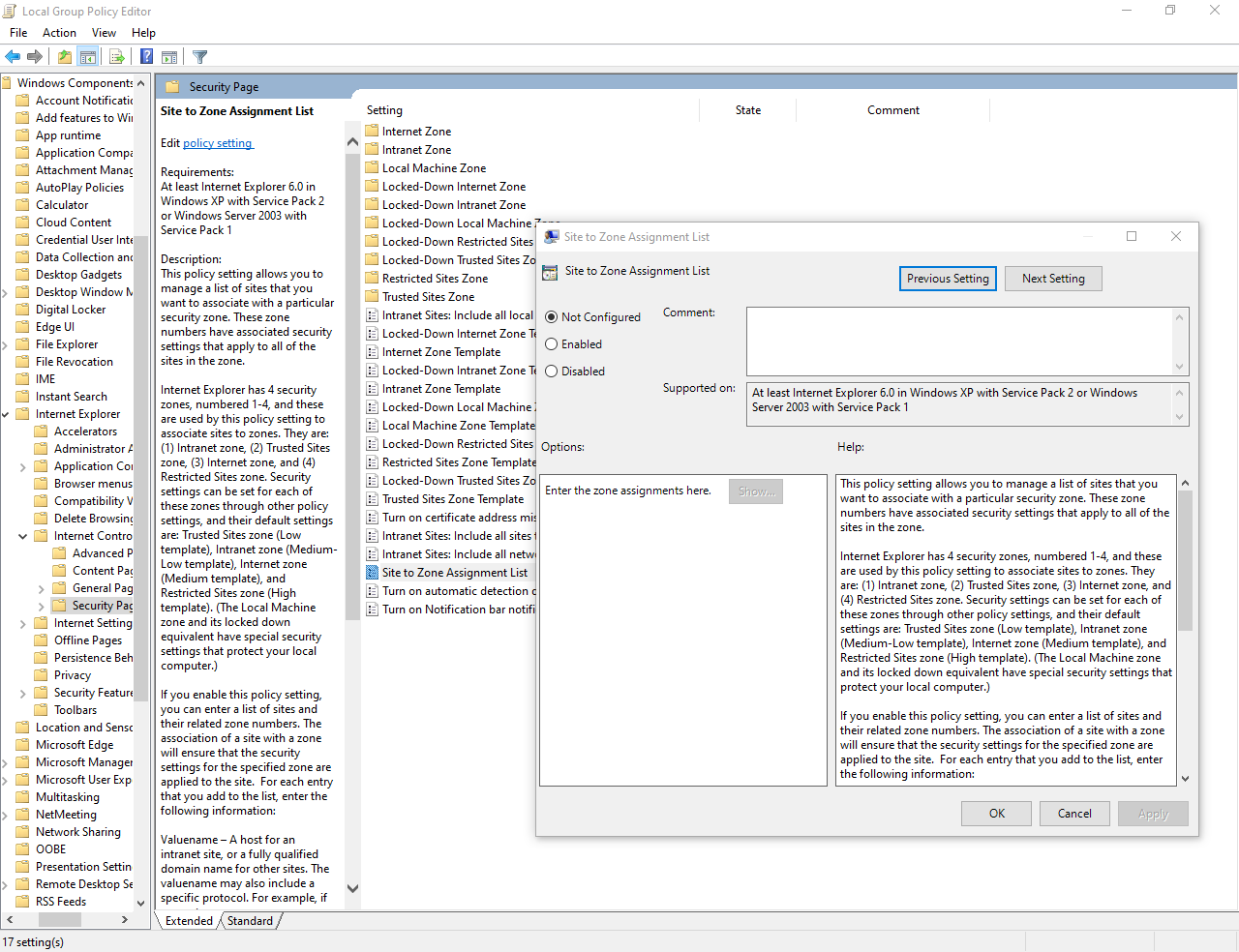
When you open files from the network, **they're subjected to Internet Explorer's security model** — yes, even in 2025. This **Trusted Sites move reclassifies that location as "safe,"** so you:

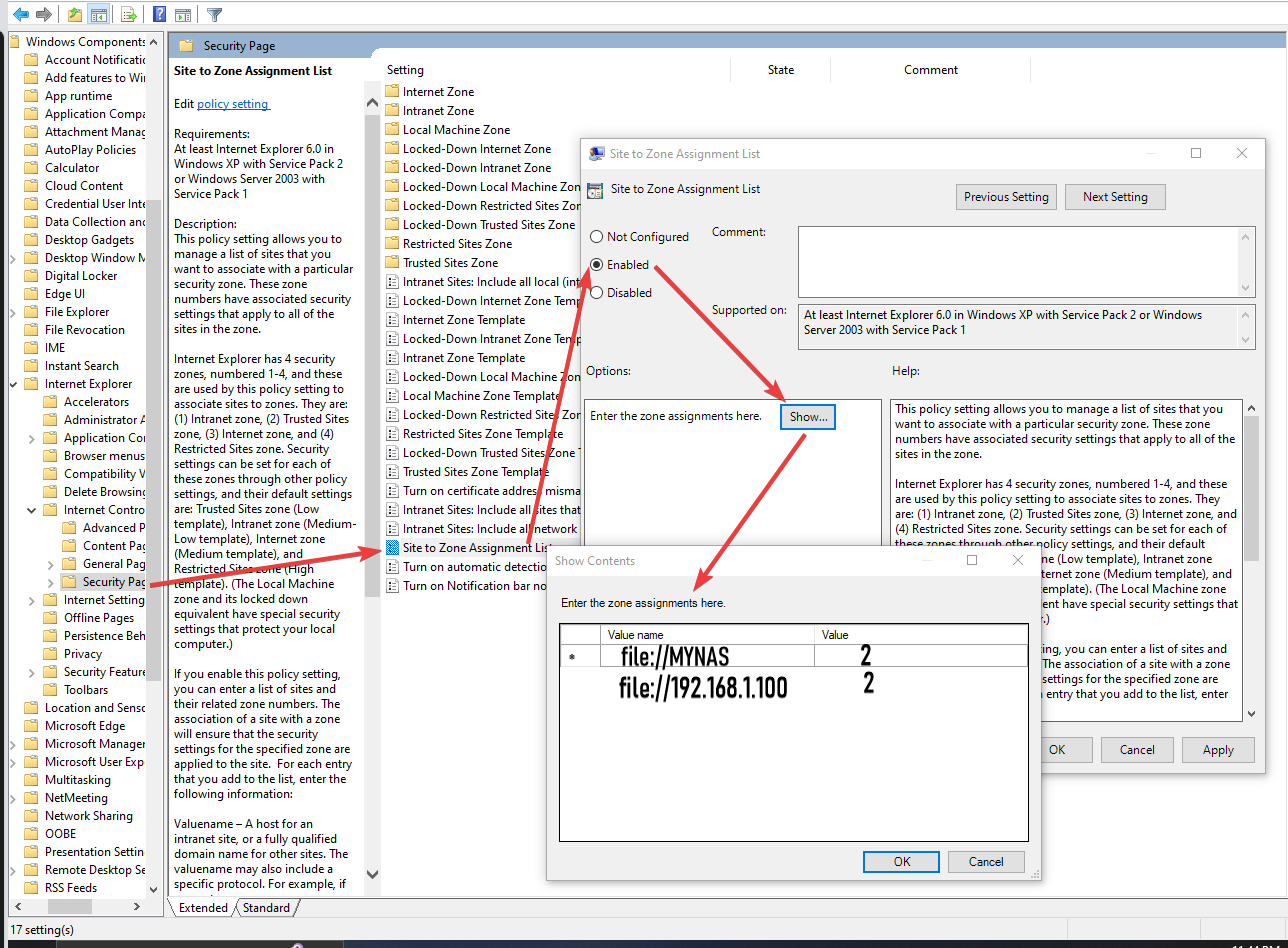
* Avoid UAC nags
* Can run .bat, .exe, .ps1, .vbs off that network
* Avoid the "This file came from another computer..." warning
* And your scripts/policies won't fail silently

**💡** Pro-Tip with GPO (Group Policy)

If you're in a domain or want to **automate this on many machines**, you can do it via Group Policy:

**User Configuration** **>** **Administrative Templates** **> Windows Components** **>** **Internet Explorer** **>** **Internet Control Panel** **>** **Security Page** **>** **Site to Zone Assignment List**





Let’s get practical and map my VM drive using both methods…

**[videos will be available later…]**

**🧨**Vista Command Line Legacy

**🟠** "Vista Doesn't Support Old Commands"

* This was a *big deal* back then — commands like choice, edlin, debug, command.com either got changed, deprecated, or removed.
* **Windows 10 Status?**  
  Most of this isn’t relevant anymore. You're *way past Vista*, and:
  + **choice** is back and standardized (with /C, /M, etc.).
  + **bcdedit** is now *permanent* — you won’t be using bootcfg ever again.
  + Many old 16-bit or DOS-based commands are **dead**. Long buried.

**✅ Takeaway for Win10:** If you run into batch file problems, don’t assume it’s a command issue — **it’s probably UAC/permissions**.

**🧪** Batch Files Breaking? Probably Permissions

“Always try to start the batch file by right-clicking it and choosing Run as Administrator…”

Still **100% true** in Windows 10. The most common mistake when a script “fails” is assuming the logic is broken. It’s not. UAC just silently denies a write, registry edit, or access.

**✅ On Win10:**  
Always test elevated if your script touches:

* Registry
* Program Files or Windows directory
* System32
* Service or scheduled task configuration

**🛠** Command Line = Underrated Power Tool

He’s preaching now. The idea that “people fiddle with GUIs while you blast through CMD” — **yup, that still slaps in 2025**.

**You're not just learning command-line basics.** You're learning:

* How to script away UI fluff
* How to automate permission hell
* How to make ancient tools work on modern systems

**💡** Final Checklist (Updated for Win10):

1. **bcdedit** – BCDEdit is the modern tool for boot configuration (not bootcfg).
2. **Learn both cmd + PowerShell and when to switch.** Some tools still only work in CMD. PowerShell is an increasingly important command-line shell to learn alongside Command Prompt. Both are crucial for reverse engineering and malware analysis.
3. **UAC is still king** – Some scripts fail silently without elevation.
4. **Deprecated commands won’t come back** – Avoid nostalgia traps.
5. **Win10 still doesn’t promote CLI** (CommandLine interface) – That’s your edge, not your obstacle.
6. **Set up right-click “Open Command Prompt Here as Admin” via registry** – It’s worth it.

**🧭** Ready for Chapter 2?

You're moving into:

* dir, xcopy, robocopy, del, attrib, etc.
* Command-line **text editors**
* Your first **registry tweaks**
* Command-line **backup automation**

Basically, we’re about to start **actually building tools**, not just understanding how Windows gets in our way.

✨**Let’s get it moving superstar!!** ✨😤😤