Automating Screenshot Cleanup with PowerShell

So while making adjustments to my machine trying to squeeze every ounce of performance out of it, I noticed that my screenshots folder was chock full of screenshots. After deleting them, I felt that automating/scheduling the clearing of the screenshots folder and then emptying the trash bin would be an interesting way for me to learn more about my system and gain experience.

Not being at all familiar with PowerShell scripting, I utilized my AI prompt engineering skills to bring this concept to life. Here I break down what I did and there's a template too if you'd like to do the same thing.

(Desktop Shortcut for the clear screenshots and empty trash .cmd)

1. An idea:

I noticed that my Screenshots folder was constantly filling up with old images, cluttering my system and taking unnecessary space. Instead of deleting them manually every time, I wanted a way to automate the cleanup process so it would happen on its own.

2. Planning the solution - I mapped out what the automation should do:

- Locate the Screenshots folder whether it is stored locally or in OneDrive
- Move all files to the Recycle Bin
- Wait a few minutes before emptying the bin
- Run automatically on a schedule or manually with a shortcut

I also wanted the script to run quietly in the background without the PowerShell window staying open.

3. Writing and refining the PowerShell script

Since I did not know PowerShell scripting, I used my prompt engineering background to leverage AI to streamline and effectively execute a working script through trial and error. I started by creating a .ps1 file that could automatically locate my Screenshots folder, whether it was stored locally or in OneDrive.

To safely move files without permanently deleting them, I used the Microsoft.VisualBasic.FileIO.FileSystem class, specifically the DeleteFile and DeleteDirectory methods with the RecycleOption.SendToRecycleBin parameter.

This approach mimics what happens when you delete something manually in Windows Explorer by sending it to the Recycle Bin instead of removing it immediately. Through several iterations, AI helped me debug syntax errors, organize logic, and ensure the script worked reliably from start to finish.

4. Adding a timer delay

To prevent instant deletion, I added a five minute delay before the Recycle Bin empties. This delay gives me time to recover something if I accidentally delete an important screenshot.

5. Installing the BurntToast module

To make the automation more user friendly, I added visual toast notifications using the BurntToast module. This module is installed through PowerShell and allows scripts to display modern notification bubbles on Windows.

Steps to install and use it:

1. Open PowerShell as Administrator.
2. Run Install-Module BurntToast -Scope CurrentUser and press Enter.
3. If prompted, confirm the installation from the PowerShell Gallery.
4. Add Import-Module BurntToast to the top of your script to enable it.
5. Test it by running New-BurntToastNotification -Text "Hello from PowerShell" to make sure it works.

6. Hiding the console window

To keep everything running silently, I used a small Windows API call to hide the PowerShell console window once the script starts. This made the automation feel smooth and invisible in the background.

Steps to do this:

1. Add the following code near the top of the script, right after any module imports:

```
Add-Type -Name Win32 -Namespace Console -MemberDefinition '
  [DllImport("user32.dll")] public static extern bool ShowWindow(IntPtr hWnd, int nCmdShow);
  [DllImport("kernel32.dll")] public static extern IntPtr GetConsoleWindow();
'
$hwnd = [Console.Win32]::GetConsoleWindow()
[Console.Win32]::ShowWindow($hwnd, 0)
```

2. This code tells Windows to hide the PowerShell console window (value 0 means hidden).
3. Save and run the script to confirm that the PowerShell window disappears after execution begins.
4. If you ever need to debug, temporarily change the last line to 5 to make the console visible again.

7. Scheduling automatic runs

I used Windows Task Scheduler to have the script run automatically every night. This lets Windows take care of the cleanup without me needing to launch it manually.
Steps to schedule it:

1. Open Task Scheduler from the Start menu.
2. Select “Create Task” in the right panel.
3. Under the General tab, name it something like Clear Screenshots.
4. Go to the Triggers tab and select “New,” then choose Daily and set a preferred time.
5. Under the Actions tab, select “New,” then browse to the PowerShell executable (usually C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe).
6. In the “Add arguments” field, enter -ExecutionPolicy Bypass -File "C:\Path\To\YourScript.ps1".
7. Save and test the task to make sure it runs correctly.

8. Creating a desktop shortcut

For times when I want to manually run the cleanup, I created a desktop shortcut that launches the script with a single click.
Steps to create it:

1. Right click on the desktop and select New → Shortcut.
2. In the location field, paste:
```
powershell.exe -ExecutionPolicy Bypass -File "C:\Path\To\YourScript.ps1"
```
3. Click Next, give it a name such as Clear Screenshots, and click Finish.
4. (Optional) Right click the shortcut, choose Properties, then Change Icon to select an icon that matches the theme of the project.

9. Testing and refining

Finally, I tested the script multiple times to make sure each function worked correctly, checking that files were moved properly, the delay was accurate, the notifications appeared, and the Recycle Bin emptied successfully.

Tips and Troubleshooting

• If toast notifications do not appear, reopen PowerShell as Administrator and reinstall BurntToast using Install-Module BurntToast -Force.

• If Task Scheduler does not run the script, ensure the PowerShell path and script location are correct and that the task is set to run with the highest privileges.
• To debug hidden windows, set ShowWindow to value 5 in the script so you can see output messages.

Script Template.

```
# --- Clear-Screenshots-Toast.ps1 ---
# Move everything in Screenshots to Recycle Bin, show a 5-minute toast
countdown,
# then empty the Recycle Bin. Console hides during the wait.

$ErrorActionPreference = 'Stop'
$ProgressPreference    = 'Continue'

# Transcript log (optional)
$log = Join-Path $env:TEMP "ClearScreenshots_$(Get-Date -Format
yyyyMMdd_HHmmss).log"
try { Start-Transcript -Path $log -ErrorAction SilentlyContinue | Out-Null }
catch {}

function Get-ScreenshotsPath {
    $candidates = @(
        (Join-Path $env:USERPROFILE "Pictures\Screenshots"),
        (Join-Path $env:USERPROFILE "OneDrive\Pictures\Screenshots")
    )
    # OneDrive business/org variants (e.g., "OneDrive - Contoso")
    $oneDriveRoots = Get-ChildItem -Path (Join-Path $env:USERPROFILE
"OneDrive*") -Directory -ErrorAction SilentlyContinue
    foreach ($root in $oneDriveRoots) {
        $candidates += (Join-Path $root.FullName "Pictures\Screenshots")
    }
    foreach ($p in $candidates | Get-Unique) {
        if (Test-Path $p) { return $p }
    }
    return $null
}
```

```
try {
    Add-Type -AssemblyName Microsoft.VisualBasic

    $ScreenshotsPath = Get-ScreenshotsPath
    if (-not $ScreenshotsPath) {
        throw "Could not locate a Screenshots folder. Set `$ScreenshotsPath
manually."
    }

    # --- Move files to Recycle Bin ---
    $itemsBefore = Get-ChildItem -Path $ScreenshotsPath -Force -ErrorAction
SilentlyContinue
    Write-Host "Using Screenshots folder: $ScreenshotsPath"
    Write-Host ("Items found before: {0}" -f $itemsBefore.Count)

    foreach ($f in (Get-ChildItem -Path $ScreenshotsPath -File -Force -
ErrorAction SilentlyContinue)) {
        try {

[Microsoft.VisualBasic.FileIO.FileSystem]::DeleteFile($f.FullName,'OnlyErrorDi
alogs','SendToRecycleBin')
            Write-Host "Moved file to Recycle Bin: $($f.Name)"
        } catch {
            Write-Host "FAILED file: $($f.FullName) ->
$($_.Exception.Message)" -ForegroundColor Red
        }
    }

    foreach ($d in (Get-ChildItem -Path $ScreenshotsPath -Directory -Force -
ErrorAction SilentlyContinue)) {
        try {

[Microsoft.VisualBasic.FileIO.FileSystem]::DeleteDirectory($d.FullName,'OnlyEr
rorDialogs','SendToRecycleBin')
            Write-Host "Moved folder to Recycle Bin: $($d.Name)"
        } catch {
            Write-Host "FAILED folder: $($d.FullName) ->
$($_.Exception.Message)" -ForegroundColor Red
        }
    }

    # --- Load BurntToast for toasts (install once if missing) ---
    $toastAvailable = $true
    try {
        if (-not (Get-Module -ListAvailable -Name BurntToast)) {
            Install-Module BurntToast -Scope CurrentUser -Force -ErrorAction
Stop
        }
        Import-Module BurntToast -ErrorAction Stop
    } catch {
        $toastAvailable = $false
        Write-Host "BurntToast not available: $($_.Exception.Message)" -
ForegroundColor Yellow
    }

    # --- Hide the console BEFORE the wait ---
    Add-Type -Name Win32 -Namespace Console -MemberDefinition '
      [DllImport("user32.dll")] public static extern bool ShowWindow(IntPtr
hWnd, int nCmdShow);
      [DllImport("kernel32.dll")] public static extern IntPtr
GetConsoleWindow();
    '
    $hwnd = [Console.Win32]::GetConsoleWindow()
    [Console.Win32]::ShowWindow($hwnd, 0)   # 0 = hide

    # --- 5-minute countdown using toasts (fallback = silent sleep) ---
    $seconds = 300
    if ($toastAvailable) {
        for ($left = $seconds; $left -gt 0; $left -= 60) {
            $mins = [math]::Ceiling($left / 60)
            try { New-BurntToastNotification -Text "Emptying Recycle Bin in
$mins minute$((if($mins -gt 1){'s'} else {''}))..." } catch {}
            Start-Sleep -Seconds ([math]::Min(60, $left))
        }
        try { New-BurntToastNotification -Text "Emptying Recycle Bin now..." }
catch {}
    } else {
        # No toast support—just wait silently
        Start-Sleep -Seconds $seconds
    }

    # --- Empty Recycle Bin ---
    Clear-RecycleBin -Force -ErrorAction SilentlyContinue

    if ($toastAvailable) {
        try { New-BurntToastNotification -Text "Recycle Bin emptied ✅" }
catch {}
    }

    # (No Read-Host here—window is hidden.)
}
catch {
    # If the window is hidden, a toast is the only thing user sees on error
    try {
        if (Get-Module -ListAvailable -Name BurntToast) {
            Import-Module BurntToast -ErrorAction SilentlyContinue
            New-BurntToastNotification -Text "Clear Screenshots error",
"$($_.Exception.Message)"
        }
    } catch {}
}
finally {
    try { Stop-Transcript | Out-Null } catch {}
}
```