# <#

# Sentinel "Manual Override" Upgrade Script (v2.4)

This script will be run by your current, buggy watcher (v2.2).

Its only purpose is to overwrite 'Sentinel.py' with the new, fully-fixed

v2.4 code.

The v2.4 code includes:

1. Fix for the 'proj-b6cc' parser bug.
2. Persistent database ('sentinel\_processed.json') to track run files.
3. Full .docx support (which you are currently using).

# AFTER YOU RUN THIS SCRIPT, YOU MUST MANUALLY RESTART THE WATCHER.

#>

# --- 1. v2.4 Configuration ---

Project-ID: proj

$SentinelPyPath = "C:\Dev\sentinel\Sentinel.py"

$CommitMessage = "feat(sentinel): Upgrade to v2.4 - Add persistent DB and fix parser"

# --- 2. The Complete v2.4 Sentinel.py Code ---

# This here-string contains the ***entire*** new Python script.

$NewSentinelPyCode = @"

import sys

import os

import argparse

import re

import time

import subprocess

import ast

import io

import json

import uuid

# --- Try to import GUI libraries (for 'register' command) ---

try:

import tkinter as tk

from tkinter import filedialog

TKINTER\_INSTALLED = True

except ImportError:

TKINTER\_INSTALLED = False

# --- Try to import watcher libraries ---

try:

from watchdog.observers import Observer

from watchdog.events import FileSystemEventHandler

WATCHDOG\_INSTALLED = True

except ImportError:

WATCHDOG\_INSTALLED = False

# --- Try to import Google API libraries ---

try:

from google.auth.transport.requests import Request

from google.oauth2.credentials import Credentials

from google\_auth\_oauthlib.flow import InstalledAppFlow

from googleapiclient.discovery import build

from googleapiclient.http import MediaIoBaseDownload

from googleapiclient.errors import HttpError

GOOGLE\_API\_INSTALLED = True

except ImportError:

GOOGLE\_API\_INSTALLED = False

# --- Try to import DOCX library ---

try:

import docx

DOCX\_INSTALLED = True

except ImportError:

DOCX\_INSTALLED = False

# --- Configuration (v2.4) ---

SENTINEL\_HOME\_DIR = os.path.dirname(os.path.abspath(file))

CONFIG\_FILE = os.path.join(SENTINEL\_HOME\_DIR, "sentinel\_config.json")

PROCESSED\_FILES\_DB = os.path.join(SENTINEL\_HOME\_DIR, "sentinel\_processed.json") # v2.4: Persistent DB

TEMP\_SCRIPT\_NAME = "\_current\_patch.ps1"

POLL\_INTERVAL\_SECONDS = 30

TEMP\_DOCX\_DOWNLOAD = os.path.join(SENTINEL\_HOME\_DIR, "\_temp\_patch.docx")

# Google Drive:

SCOPES = ["https://www.googleapis.com/auth/drive"]

CREDENTIALS\_FILE = os.path.join(SENTINEL\_HOME\_DIR, "credentials.json")

TOKEN\_FILE = os.path.join(SENTINEL\_HOME\_DIR, "token.json")

DOCX\_MIME\_TYPE = "application/vnd.openxmlformats-officedocument.wordprocessingml.document"

# ---------------------

# --- v2.4 DB Helper Functions ---

def load\_processed\_files():

"""Loads the set of processed file IDs from the JSON DB."""

if not os.path.exists(PROCESSED\_FILES\_DB):

return set()

try:

with open(PROCESSED\_FILES\_DB, 'r', encoding='utf-8') as f:

return set(json.load(f)) # Load list and convert to set

except Exception:

print(f"Warning: Could not read {PROCESSED\_FILES\_DB}. Starting with empty set.", file=sys.stderr)

return set()

def save\_processed\_files(processed\_set):

"""Saves the set of processed file IDs to the JSON DB."""

try:

with open(PROCESSED\_FILES\_DB, 'w', encoding='utf-8') as f:

json.dump(list(processed\_set), f, indent=4) # Convert set to list for saving

except Exception as e:

print(f"[Watcher] Error: Could not save processed files list: {e}", file=sys.stderr)

# --- End v2.4 DB ---

def load\_config():

if not os.path.exists(CONFIG\_FILE):

return {}

try:

with open(CONFIG\_FILE, 'r', encoding='utf-8') as f:

return json.load(f)

except json.JSONDecodeError:

print(f"Warning: Config file {CONFIG\_FILE} is corrupted. Returning empty config.", file=sys.stderr)

return {}

def save\_config(config\_data):

try:

with open(CONFIG\_FILE, 'w', encoding='utf-8') as f:

json.dump(config\_data, f, indent=4)

return True

except Exception as e:

print(f"Error saving config file: {e}", file=sys.stderr)

return False

# ==============================================================================

# --- "LOCAL WATCHER" (SERVICE) LOGIC (v2.4) ---

# ==============================================================================

def start\_local\_watcher():

if not WATCHDOG\_INSTALLED:

print("Error: 'watchdog' is required. Please run: pip install watchdog", file=sys.stderr)

sys.exit(1)

config = load\_config()  
if not config:  
 print("Error: No projects registered. Run 'python Sentinel.py register' first.", file=sys.stderr)  
 sys.exit(1)  
   
# v2.4: Load persistent DB  
processed\_file\_paths = load\_processed\_files() # Using path for local, ID for drive  
  
print("Local Watcher scanning for registered projects:")  
for proj\_id, path in config.items():  
 print(f" - {proj\_id}: {path}")  
  
class LocalPatchHandler(FileSystemEventHandler):  
 def on\_created(self, event):  
 if event.is\_directory: return  
 filepath = os.path.normpath(event.src\_path)  
 filename = os.path.basename(filepath)  
   
 # v2.4: Check persistent DB  
 if filepath in processed\_file\_paths:  
 return # We've already run this file  
  
 if filename.endswith(".txt") and filename.startswith("SentScript-"):  
 try:  
 # --- FIXED v2.4 PARSER LOGIC ---  
 parts = filename.split('-')  
 if len(parts) < 4:   
 print(f"\n[Local Watcher] Invalid filename format (not enough parts): {filename}. Ignoring.", flush=True)  
 return  
 project\_id = f"{parts[1]}-{parts[2]}" # Re-combine 'proj' and 'b6cc'  
 # --- END FIXED LOGIC ---  
   
 config = load\_config()  
   
 if project\_id not in config:  
 print(f"\n[Local Watcher] Detected file for unknown project ID: {project\_id}. Ignoring.", flush=True)  
 return  
   
 target\_project\_path = config[project\_id]  
 print(f"\n\n--- [Local Watcher] New Patch File Detected: {filename} ---", flush=True)  
 print(f"--- Target Project: {project\_id} ({target\_project\_path}) ---", flush=True)  
  
 try:  
 with open(filepath, 'r', encoding='utf-8') as f:  
 script\_content = f.read()  
 except UnicodeDecodeError:  
 print(f"[Watcher] Warning: UTF-8 decode failed. Retrying with 'latin-1'...", flush=True)  
 with open(filepath, 'r', encoding='latin-1') as f:  
 script\_content = f.read()  
   
 # Run the patch  
 verify\_and\_run\_patch(script\_content, filename, target\_project\_path)  
   
 # v2.4: Mark as processed and save DB  
 processed\_file\_paths.add(filepath)  
 save\_processed\_files(processed\_file\_paths)  
  
 except Exception as e:  
 print(f"[Local Watcher] Error processing file: {e}. Ignoring.", flush=True)  
  
observer = Observer()  
for path in config.values():  
 if os.path.exists(path):  
 observer.schedule(LocalPatchHandler(), path, recursive=False)  
 else:  
 print(f"Warning: Path not found for project. Not watching: {path}", file=sys.stderr)  
  
print("==================================================")  
print("✅ Sentinel 'Local Watcher' Service Started (v2.4)")  
print(f"Loaded {len(processed\_file\_paths)} already-processed file paths.")  
print(f"Watching for new 'SentScript-ID-\*.txt' files in all registered project folders.")  
print("Press CTRL+C to stop the watcher.")  
print("==================================================")  
  
observer.start()  
try:  
 while True:  
 time.sleep(1)  
except KeyboardInterrupt:  
 observer.stop()  
 print("\n[Local Watcher] Service stopped by user.")  
observer.join()

# ==============================================================================

# --- "GOOGLE DRIVE WATCHER" (SERVICE) LOGIC (v2.4) ---

# ==============================================================================

def get\_drive\_service():

creds = None

if os.path.exists(TOKEN\_FILE):

creds = Credentials.from\_authorized\_user\_file(TOKEN\_FILE, SCOPES)

if not creds or not creds.valid:

if creds and creds.expired and creds.refresh\_token:

creds.refresh(Request())

else:

if not os.path.exists(CREDENTIALS\_FILE):

print(f"FATAL ERROR: '{CREDENTIALS\_FILE}' not found.", file=sys.stderr)

sys.exit(1)

flow = InstalledAppFlow.from\_client\_secrets\_file(CREDENTIALS\_FILE, SCOPES)

creds = flow.run\_local\_server(port=0)

with open(TOKEN\_FILE, 'w') as token:

token.write(creds.to\_json())

return build('drive', 'v3', credentials=creds)

def read\_docx\_text(filepath):

"""Uses python-docx to read the text from a .docx file."""

try:

document = docx.Document(filepath)

full\_text = []

for para in document.paragraphs:

full\_text.append(para.text)

return "\n".join(full\_text)

except Exception as e:

print(f"[Watcher] Error: Failed to read .docx file: {e}", file=sys.stderr)

return None

def start\_drive\_watcher():

if not GOOGLE\_API\_INSTALLED or not DOCX\_INSTALLED:

print("Error: Missing required libraries for Drive Watcher.", file=sys.stderr)

if not GOOGLE\_API\_INSTALLED:

print("Please run: pip install google-api-python-client google-auth-httplib2 google-auth-oauthlib", file=sys.stderr)

if not DOCX\_INSTALLED:

print("Please run: pip install python-docx", file=sys.stderr)

sys.exit(1)

print("Authenticating with Google Drive...")  
try:  
 service = get\_drive\_service()  
 print("Authentication successful.")  
except Exception as e:  
 print(f"Failed to authenticate with Google Drive: {e}", file=sys.stderr)  
 sys.exit(1)  
  
# --- v2.4: Load persistent list of processed files ---  
processed\_file\_ids = load\_processed\_files()  
  
print("==================================================")  
print("✅ Sentinel 'Google Drive Watcher' Service Started (v2.4)")  
print(f"Loaded {len(processed\_file\_ids)} already-processed file IDs.")  
print(f"Polling for new 'SentScript-ID-\*.docx' files every {POLL\_INTERVAL\_SECONDS} seconds.")  
print("Press CTRL+C to stop the watcher.")  
print("==================================================")  
  
try:  
 while True:  
 try:  
 # --- v2.4: Still query by name, it's more efficient ---  
 results = service.files().list(  
 q=f"mimeType='{DOCX\_MIME\_TYPE}' and name starts with 'SentScript-'",  
 pageSize=20,  
 orderBy="createdTime desc",  
 fields="files(id, name, createdTime)"  
 ).execute()  
   
 items = results.get('files', [])  
 new\_files\_found = 0  
  
 if items:  
 config = load\_config()  
 for item in reversed(items):  
 file\_id = item['id']  
 filename = item['name']  
   
 # --- v2.4: Skip if we've already processed it ---  
 if file\_id in processed\_file\_ids:  
 continue  
  
 new\_files\_found += 1  
   
 try:  
 # --- FIXED v2.4 PARSER LOGIC ---  
 clean\_name = os.path.splitext(filename)[0]  
 parts = clean\_name.split('-')  
 if len(parts) < 4:   
 print(f"\n[Drive Watcher] Invalid filename format (not enough parts): {filename}. Ignoring.", flush=True)  
 processed\_file\_ids.add(file\_id) # Mark as processed  
 save\_processed\_files(processed\_file\_ids)  
 continue  
 project\_id = f"{parts[1]}-{parts[2]}" # Re-combine 'proj' and 'b6cc'  
 # --- END FIXED LOGIC ---  
 except IndexError:  
 print(f"\n[Drive Watcher] Invalid filename format (no ID): {filename}. Ignoring.", flush=True)  
 processed\_file\_ids.add(file\_id)  
 continue  
   
 if project\_id not in config:  
 # --- v2.4: Allow 'proj' ID for this one patch ---  
 if project\_id == "proj" and "proj" in config:  
 pass # Allow this specific upgrade script  
 else:  
 print(f"\n[Drive Watcher] Detected file for unknown project ID: {project\_id}. Ignoring.", flush=True)  
 processed\_file\_ids.add(file\_id)  
 save\_processed\_files(processed\_file\_ids)  
 continue  
   
 target\_project\_path = config[project\_id]  
 print(f"\n--- [Drive Watcher] New Patch File Detected: {filename} (ID: {file\_id}) ---", flush=True)  
 print(f"--- Target Project: {project\_id} ({target\_project\_path}) ---", flush=True)  
   
 try:  
 # 1. Download .docx file  
 request = service.files().get\_media(fileId=file\_id)  
 fh = io.FileIO(TEMP\_DOCX\_DOWNLOAD, 'wb')  
 downloader = MediaIoBaseDownload(fh, request)  
 done = False  
 while done is False:  
 status, done = downloader.next\_chunk()  
 fh.close()  
   
 # 2. Read .docx file  
 print(f"[Watcher] Download complete. Parsing .docx file...", flush=True)  
 script\_content = read\_docx\_text(TEMP\_DOCX\_DOWNLOAD)  
   
 if os.path.exists(TEMP\_DOCX\_DOWNLOAD):  
 os.remove(TEMP\_DOCX\_DOWNLOAD) # Clean up temp file  
  
 if script\_content is None:  
 raise Exception("Failed to read text from .docx file.")  
   
 # 3. v2.4: Content check  
 if not script\_content.lstrip().startswith("<#"):  
 print(f"[Watcher] File {filename} is not a SentScript (missing '<#'). Ignoring.", flush=True)  
 processed\_file\_ids.add(file\_id) # Mark as processed  
 save\_processed\_files(processed\_file\_ids)  
 continue  
  
 # 4. Verify and Run  
 if verify\_and\_run\_patch(script\_content, filename, target\_project\_path):  
 print(f"[Watcher] Patch script ran successfully for {filename}.", flush=True)  
 else:  
 print(f"[Watcher] Patch failed or aborted for {filename}.", flush=True)  
  
 # 5. --- v2.4: Mark as processed and save ---  
 processed\_file\_ids.add(file\_id)  
 save\_processed\_files(processed\_file\_ids)  
 # We no longer delete the file from Drive  
  
 except HttpError as e:  
 print(f"[Drive Watcher] Error processing file {filename}: {e}", flush=True)  
 except Exception as e:  
 print(f"[Drive Watcher] A critical error occurred processing {filename}: {e}", flush=True)  
   
 if new\_files\_found == 0:  
 print(f"[{time.ctime()}] No new .docx files found. Sleeping...", end="\r", flush=True)  
 else:  
 print(f"[{time.ctime()}] All new files processed. Sleeping...", end="\r", flush=True)  
   
 except HttpError as e:  
 print(f"\n[Drive Watcher] API Error: {e}. Retrying...", flush=True)  
 except Exception as e:  
 print(f"\n[Drive Watcher] An unexpected error occurred: {e}", flush=True)  
   
 time.sleep(POLL\_INTERVAL\_SECONDS)  
   
except KeyboardInterrupt:  
 print("\n[Drive Watcher] Service stopped by user.")

# ==============================================================================

# --- "ROBOT SURGEON" (TOOL) & SHARED LOGIC ---

# ==============================================================================

def verify\_and\_run\_patch(script\_content, source\_filename, target\_project\_path):

if not script\_content.strip():

print(f"[Watcher] File '{source\_filename}' is empty. Ignoring.", flush=True)

return False

print(f"--- VERIFY PATCH FOR: {source\_filename} ---")  
print(script\_content)  
print("--- END OF PATCH SCRIPT ---")  
  
try:  
 choice = input(f"Do you approve and want to RUN this patch? (y/n): ")  
except EOFError:  
 choice = 'n'  
  
if choice.lower().strip() != 'y':  
 print("[Watcher] User aborted.", flush=True)  
 return False  
  
print("[Watcher] User approved. Executing patch...", flush=True)  
temp\_script\_path = os.path.join(target\_project\_path, TEMP\_SCRIPT\_NAME)  
  
try:  
 with open(temp\_script\_path, 'w', encoding='utf-8') as f:  
 f.write(script\_content)  
   
 subprocess.run(  
 ["powershell.exe", "-ExecutionPolicy", "Bypass", "-File", TEMP\_SCRIPT\_NAME],  
 check=True, shell=True, cwd=target\_project\_path  
 )  
 print("[Watcher] Patch script executed successfully.", flush=True)  
 return True  
  
except Exception as e:  
 print(f"[Watcher] ERROR: The patch script failed to run: {e}", flush=True)  
 return False  
finally:  
 if os.path.exists(temp\_script\_path):  
 os.remove(temp\_script\_path)

def get\_block\_regex(block\_name, file\_extension):

escaped\_name = re.escape(block\_name)

if file\_extension == ".py":

start\_sentinel = rf"# --- BLOCK: {escaped\_name} ---"

end\_sentinel = rf"# --- ENDBLOCK: {escaped\_name} ---"

else:

start\_sentinel = rf"<!-- BLOCK: {escaped\_name} -->"

end\_sentinel = rf"<!-- ENDBLOCK: {escaped\_name} -->"

return re.compile(f"(?s)({re.escape(start\_sentinel)})(.\*)({re.escape(end\_sentinel)})")

def patch\_file(filepath, block\_name, new\_content):

if not os.path.exists(filepath):

print(f"Error: File not found: {filepath}", file=sys.stderr)

return False

try:

try:

with open(filepath, 'r', encoding='utf-8') as f:

file\_content = f.read()

except UnicodeDecodeError:

print(f"[Patcher] Warning: UTF-8 decode failed on {filepath}. Retrying with 'latin-1'...", file=sys.stderr)

with open(filepath, 'r', encoding='latin-1') as f:

file\_content = f.read()

except Exception as e:

print(f"Error reading file: {e}", file=sys.stderr)

return False

\_, file\_extension = os.path.splitext(filepath)  
regex\_pattern = get\_block\_regex(block\_name, file\_extension)  
  
if not regex\_pattern.search(file\_content):  
 print(f"Error: Could not find sentinels for block '{block\_name}' in {filepath}", file=sys.stderr)  
 return False  
   
match = regex\_pattern.search(file\_content)  
clean\_old\_content = re.sub(r'\s+', ' ', match.group(2).strip())  
clean\_new\_content = re.sub(r'\s+', ' ', new\_content.strip())  
  
if clean\_old\_content == clean\_new\_content:  
 print(f"Validation Failed: Patch content for '{block\_name}' is identical to existing code.", file=sys.stderr)  
 return True   
   
replacement = f"\1\n{new\_content}\n\3"  
new\_file\_content = regex\_pattern.sub(replacement, file\_content)  
  
try:  
 with open(filepath, 'w', encoding='utf-8') as f:  
 f.write(new\_file\_content)  
except UnicodeEncodeError:  
 print(f"[Patcher] Warning: UTF-8 encode failed. Retrying with 'latin-1'...", file=sys.stderr)  
 with open(filepath, 'w', encoding='latin-1') as f:  
 f.write(new\_file\_content)  
except Exception as e:  
 print(f"Error writing to file: {e}", file=sys.stderr)  
 return False  
   
print(f"SUCCESS: Patched block '{block\_name}' in '{filepath}'")  
return True

def bootstrap\_file(filepath):

if not os.path.exists(filepath):

print(f"Error: File not found: {filepath}", file=sys.stderr)

return False

try:

try:

with open(filepath, 'r', encoding='utf-8') as f:

file\_content = f.read()

except UnicodeDecodeError:

print(f"[Bootstrapper] Warning: UTF-8 decode failed. Retrying with 'latin-1'...", file=sys.stderr)

with open(filepath, 'r', encoding='latin-1') as f:

file\_content = f.read()

except Exception as e:

print(f"Error reading file: {e}", file=sys.stderr)

return False

if "# --- BLOCK:" in file\_content or "<!-- BLOCK:" in file\_content:  
 print("Error: File already appears to be bootstrapped. Aborting.", file=sys.stderr)  
 return False  
   
try:  
 tree = ast.parse(file\_content)  
 lines = file\_content.splitlines(True)  
except Exception as e:  
 print(f"Error parsing Python file: {e}", file=sys.stderr)  
 return False  
  
for node in ast.walk(tree):  
 for child in ast.iter\_child\_nodes(node):  
 setattr(child, '\_parent', node)  
  
insertions = []  
for node in ast.walk(tree):  
 if isinstance(node, (ast.FunctionDef, ast.AsyncFunctionDef)) and hasattr(node, '\_parent') and isinstance(node.\_parent, ast.Module):  
 func\_name = node.name  
 start\_line = node.lineno  
 end\_line = node.end\_lineno  
   
 if node.decorator\_list:  
 start\_line = node.decorator\_list[0].lineno  
  
 if node.body:  
 last\_body\_item = node.body[-1]  
 end\_line = getattr(last\_body\_item, 'end\_lineno', last\_body\_item.lineno)  
  
 indent = " " \* node.col\_offset  
 insertions.append((start\_line - 1, f"{indent}# --- BLOCK: {func\_name} ---\n"))  
 insertions.append((end\_line, f"{indent}# --- ENDBLOCK: {func\_name} ---\n"))  
  
if not insertions:  
 print("No top-level functions found. Nothing to bootstrap.", file=sys.stderr)  
 return False  
   
insertions.sort(key=lambda x: x[0], reverse=True)  
new\_lines = list(lines)  
for line\_index, text\_to\_insert in insertions:  
 new\_lines.insert(line\_index, text\_to\_insert)  
new\_file\_content = "".join(new\_lines)  
  
try:  
 with open(filepath, 'w', encoding='utf-8') as f:  
 f.write(new\_file\_content)  
except UnicodeEncodeError:  
 print(f"[Bootstrapper] Warning: UTF-8 encode failed. Retrying with 'latin-1'...", file=sys.stderr)  
 with open(filepath, 'w', encoding='latin-1') as f:  
 f.write(new\_file\_content)  
except Exception as e:  
 print(f"Error writing bootstrapped file: {e}", file=sys.stderr)  
 return False  
   
print(f"SUCCESS: Bootstrapped '{filepath}' with {len(insertions)//2} blocks.")  
return True

def register\_project():

if not TKINTER\_INSTALLED:

print("Error: 'tkinter' is required for the register command.", file=sys.stderr)

return False

print("Opening folder selection dialog...")  
root = tk.Tk()  
root.withdraw()   
  
project\_path = filedialog.askdirectory(title="Select Project Folder to Register")  
root.destroy()  
  
if not project\_path:  
 print("No folder selected. Aborting.", flush=True)  
 return False  
   
project\_path = os.path.normpath(project\_path)  
config = load\_config()  
  
for proj\_id, path in config.items():  
 if path == project\_path:  
 print(f"Project already registered with ID: {proj\_id}", flush=True)  
 return True  
   
project\_id = f"proj-{uuid.uuid4().hex[:4]}"  
config[project\_id] = project\_path  
  
if save\_config(config):  
 print(f"SUCCESS: Registered project at '{project\_path}' with ID: {project\_id}", flush=True)  
 return True  
else:  
 print(f"Error: Failed to save updated config.", file=sys.stderr)  
 return False

# ==============================================================================

# --- MAIN "BOOTLOADER" ---

# ==============================================================================

def main():

parser = argparse.ArgumentParser(description="Sentinel v2.4: AI-driven multi-project patch manager.")

subparsers = parser.add\_subparsers(dest="command", required=True)  
  
watch\_parser = subparsers.add\_parser("watch", help="Start the watcher service.")  
watch\_parser.add\_argument("mode", choices=["local", "drive"], help="The type of watcher to run.")  
  
patch\_parser = subparsers.add\_parser("patch", help="Patch a block in a file. (Called by patch scripts)")  
patch\_parser.add\_argument("filepath", help="The file to patch (e.g., 'main.py')")  
patch\_parser.add\_argument("block\_name", help="The name of the block to patch (e.g., 'get\_dashboard')")  
  
bootstrap\_parser = subparsers.add\_parser("bootstrap", help="One-time setup to add sentinel markers to a file.")  
bootstrap\_parser..add\_argument("filepath", help="The file to bootstrap (e.g., 'main.py')")  
  
register\_parser = subparsers.add\_parser("register", help="Register a new project folder with Sentinel (opens GUI).")  
  
try:  
 args = parser.parse\_args()  
except SystemExit:  
 return  
   
if args.command == "watch":  
 if args.mode == "local":  
 start\_local\_watcher()  
 elif args.mode == "drive":  
 start\_drive\_watcher()  
   
elif args.command == "patch":  
 new\_content = sys.stdin.read()  
 if not patch\_file(args.filepath, args.block\_name, new\_content):  
 sys.exit(1)  
  
elif args.command == "bootstrap":  
 if not bootstrap\_file(args.filepath):  
 sys.exit(1)  
   
elif args.command == "register":  
 if not register\_project():  
 sys.exit(1)

if name == "main":

main()

"@

# --- 3. Overwrite the file ---

Write-Host "--- 1. Stopping any running Sentinel (just in case) ---"

# This is a soft check, won't stop a separate terminal

Stop-Process -Name "python" -ErrorAction SilentlyContinue

Write-Host "--- 2. Overwriting $SentinelPyPath with v2.4 code ---" -ForegroundColor Yellow

try {

Set-Content -Path $SentinelPyPath -Value $NewSentinelPyCode -Encoding UTF8 -ErrorAction Stop

Write-Host "SUCCESS: Sentinel.py has been upgraded to v2.4." -ForegroundColor Green

}

catch {

Write-Error "Failed to overwrite $SentinelPyPath:"

Write-Error $\_

exit 1

}

# --- 4. Commit Changes ---

Write-Host "--- 3. Committing v2.4 upgrade to Git ---" -ForegroundColor Yellow

try {

git add $SentinelPyPath

git commit -m $CommitMessage

Write-Host "SUCCESS: v2.4 upgrade committed." -ForegroundColor Green

}

catch {

Write-Error "Failed to commit to Git:"

Write-Error $\_

exit 1

}

Write-Host "======================================================="

Write-Host "✅ SENTINEL UPGRADE COMPLETE ✅"

Write-Host "The 'Sentinel.py' file is now v2.4."

Write-Host "This script did not restart the watcher."

Write-Host "IMPORTANT: You MUST manually RESTART your watcher service."

Write-Host "(e.g., python Sentinel.py watch drive)"

Write-Host "======================================================="