===================================

YouTube Shorts Downloader & Uploader

===================================

This project consists of two Python scripts designed to automate the process of finding, downloading, and uploading YouTube Shorts related to a specific niche (focused on GTA VI in the provided example).

1. **downloader.py:** Searches YouTube for Shorts based on generated keywords, downloads eligible videos, generates SEO metadata (Title, Description, Tags) using Google Gemini, saves the video and metadata, and updates an Excel tracking sheet.
2. **uploader.py:** Reads the downloaded video/metadata information, uploads the videos to a specified YouTube channel using Selenium automation, schedules uploads at configurable intervals, updates the Excel tracking sheet, and cleans up processed files.

**DISCLAIMER:**

* Automating YouTube uploads with tools like Selenium can be against YouTube's Terms of Service. Use these scripts responsibly and at your own risk. The creators are not responsible for any account suspension or other consequences.
* Downloading and re-uploading copyrighted content without permission is illegal and unethical. Ensure you have the rights or are operating under fair use principles (which can be complex) for any content you process. The script includes a basic fair use disclaimer in the generated description, but this does not guarantee legal compliance.
* Using the Google Gemini API incurs costs based on usage. Monitor your API usage and associated costs.
* YouTube Studio's website structure can change frequently, which may break the uploader script (Selenium automation). Updates to the script's selectors (XPaths, CSS Selectors) might be required periodically.

**Prerequisites**

1. **Python:** Python 3.7 or higher installed.
2. **pip:** Python package installer (usually comes with Python).
3. **Git:** (Optional) For cloning the repository easily.
4. **Firefox:** The uploader script is specifically designed for the Firefox web browser. Make sure it's installed.
5. **FFmpeg:** The downloader script (yt-dlp) requires FFmpeg for merging video and audio streams.
   * Download from the official FFmpeg website: <https://ffmpeg.org/download.html>
   * You need to either:
     + Add the directory containing ffmpeg.exe (or ffmpeg on Linux/macOS) to your system's PATH environment variable.
     + **OR** Place the ffmpeg.exe file directly in the same directory as the downloader.py script. The script looks for ffmpeg.exe in its directory by default.

**Setup**

1. **Get the Code:**
   * Clone the repository (if using Git):
   * git clone <repository\_url>
   * cd <repository\_directory>
   * Or download the downloader.py and uploader.py files into a dedicated project folder.
2. **Create a Virtual Environment (Recommended):**
   * Open a terminal or command prompt in the project folder.
   * Create the environment:
   * python -m venv venv
   * Activate the environment:
     + **Windows:** .\venv\Scripts\activate
     + **Linux/macOS:** source venv/bin/activate
3. **Install Dependencies:**
   * Create a file named requirements.txt in the project folder with the following content:
   * yt-dlp
   * google-generativeai
   * openpyxl
   * selenium
   * webdriver-manager
   * Install the packages:
   * pip install -r requirements.txt
   * *Note:* webdriver-manager will automatically download the correct geckodriver for Firefox when the uploader script runs for the first time (requires internet access).
4. **Configure API Key and Settings (config.txt):**
   * Create a file named config.txt in the project folder.
   * Add the following keys, replacing the placeholder values:

# --- Downloader Settings ---

# REQUIRED: Your Google Gemini API Key (https://aistudio.google.com/app/apikey)

API\_KEY=YOUR\_GEMINI\_API\_KEY\_HERE

# REQUIRED: Maximum number of videos to download per script run

MAX\_DOWNLOADS=50

# OPTIONAL: Max keywords to keep in cache (Default: 200 if missing)

# Keywords are generated/added until this limit is reached.

MAX\_KEYWORDS=250

# --- Uploader Settings ---

# OPTIONAL: Max videos to upload per script run (Default: 25 if missing)

MAX\_UPLOADS=10

# OPTIONAL: Default YouTube video category (Default: "Gaming" if missing)

# Find valid categories in YouTube Studio upload defaults.

UPLOAD\_CATEGORY=Gaming

# OPTIONAL: Time interval between scheduled uploads in minutes (Default: 120 if missing)

# The first video is published immediately, subsequent ones are scheduled.

SCHEDULE\_INTERVAL\_MINUTES=180

# OPTIONAL: Full path to a Firefox profile directory (Leave blank or comment out to use default profile)

# Using a profile where you are already logged into Google/YouTube can skip the manual login step.

# Find paths here: about:profiles in Firefox

# Example (Windows): PROFILE\_PATH=C:\Users\YourUser\AppData\Roaming\Mozilla\Firefox\Profiles\xxxxxxxx.default-release

# Example (Linux): PROFILE\_PATH=/home/youruser/.mozilla/firefox/xxxxxxxx.default-release

PROFILE\_PATH=

* + **Important:** Get your Gemini API Key from Google AI Studio. Keep this key secure.

1. **Define Seed Niche (niche.txt):**
   * Create a file named niche.txt in the project folder.
   * Add a *single line* defining the core niche for keyword generation. Example:
   * GTA VI
2. **Firefox Profile (Optional - Uploader):**
   * If you want the uploader to use a specific Firefox profile (e.g., one where you're already logged into YouTube), find your profile path:
     + Open Firefox.
     + Type about:profiles in the address bar.
     + Find the "Root Directory" or "Local Directory" path for the profile you want to use.
     + Copy this path and paste it as the value for PROFILE\_PATH in config.txt.
   * If PROFILE\_PATH is left blank or commented out, the script will use a default temporary Firefox profile, likely requiring you to manually log in to Google/YouTube within the browser window that Selenium opens during the first run.

**Running the Scripts**

* **Always run the downloader first** to gather videos and metadata.
* Ensure your virtual environment is activated (venv\Scripts\activate or source venv/bin/activate).
* Make sure the Excel file (shorts\_data.xlsx) is **closed** before running either script.
* **Run the Downloader:**
* python downloader.py
  + The script will:
    - Read the niche.
    - Generate/load keywords.
    - Search YouTube for each keyword.
    - Filter results for valid Shorts under 61 seconds.
    - Call the Gemini API to generate metadata (Title, Description, Tags).
    - Download the video files to shorts\_downloads/.
    - Save metadata JSON files to shorts\_metadata/.
    - Update shorts\_data.xlsx (Downloaded sheet).
    - Save cache files (playlist\_cache.json, generated\_keywords\_cache.json).
* **Run the Uploader:**
* python uploader.py
  + The script will:
    - Open Firefox (may require manual Google/YouTube login on the first run if not using a logged-in profile).
    - Check the shorts\_metadata/ folder for videos to upload.
    - For each video:
      * Navigate to YouTube Studio.
      * Fill in the Title, Description, Tags, Category, etc., using the metadata.
      * Publish the first video immediately.
      * Schedule subsequent videos based on SCHEDULE\_INTERVAL\_MINUTES.
      * Update shorts\_data.xlsx (Moves entry from Downloaded to Uploaded sheet).
      * Delete the local video file (shorts\_downloads/) and metadata file (shorts\_metadata/) after successful processing.
    - Log errors to upload\_error\_log.txt.

**File Structure**

Your\_Project\_Folder/

│

├── downloader.py # The script to download videos and metadata

├── uploader.py # The script to upload videos to YouTube

├── config.txt # Configuration file (API Key, Limits, etc.) - YOU CREATE THIS

├── niche.txt # Seed niche for keyword generation - YOU CREATE THIS

├── requirements.txt # List of Python dependencies - YOU CREATE THIS

├── ffmpeg.exe # (Optional) Place FFmpeg executable here if not in PATH

│

├── shorts\_downloads/ # Folder where videos are downloaded (Created by downloader)

│ └── video1.mp4

│ └── video2.mp4

│ └── ...

│

├── shorts\_metadata/ # Folder where metadata JSON files are saved (Created by downloader)

│ └── video1.json

│ └── video2.json

│ └── ...

│

├── shorts\_data.xlsx # Excel file tracking downloaded and uploaded videos (Created/Updated by scripts)

├── playlist\_cache.json # Caches downloaded YouTube video IDs to avoid re-downloading (Created by downloader)

├── generated\_keywords\_cache.json # Caches generated search keywords and their usage frequency (Created by downloader)

├── upload\_error\_log.txt # Logs errors encountered during the upload process (Created by uploader)

│

└── venv/ # Python virtual environment directory (Created by you)

└── ...

**Troubleshooting & Notes**

* **Uploader Fails:** The most common reason for the uploader failing is YouTube changing its website structure (HTML elements, IDs, classes). You may need to inspect the YouTube Studio page source and update the XPaths or CSS Selectors in uploader.py. Check upload\_error\_log.txt for details.
* **Manual Login:** If not using a pre-configured Firefox profile (PROFILE\_PATH), the uploader will open Firefox. You'll likely need to manually log in to your Google/YouTube account *in that specific browser window* the first time you run it. Subsequent runs should use the session cookies stored in that temporary profile (until it's automatically deleted).
* **GeckoDriver Download Failed:** webdriver-manager downloads geckodriver. If this fails, check your internet connection and firewall settings.
* **API Errors:** Check your Gemini API key is correct and your Google Cloud project has billing enabled if you exceed free tier limits. Ensure the API is enabled for your project.
* **FFmpeg Not Found:** Make sure ffmpeg.exe is in your PATH or in the same directory as downloader.py.
* **Excel File Locked:** Ensure shorts\_data.xlsx is closed before running the scripts, as they need to read and write to it.
* **Rate Limiting:** Aggressive downloading or uploading might trigger rate limits from YouTube or Google API. The scripts have some basic delays, but consider increasing them (mimic\_human\_action\_delay) if you encounter issues.