
FREE
SECRETGFX
TRAFFIC
FROM
YOUTUBE
SHORTS
AUTOMATED

INTRO

In this guide, you will learn how to get free traffic from YouTube shorts and how to automate the process with python. There are countless different ways to make various twists to this process as well as look for different niches and monetization options. The intention of this guide is to demonstrate how to achieve a fully automated & free traffic source. It is **not** intended for you to copy the methods shown as examples in this ebook. This is **not** a money-making guide, but rather a well-researched traffic method with monetization ideas. All the code shared in this guide has been tested and working as of its release. There might be changes to YouTube's algorithm in the future, so please understand that you might need to adapt the methods accordingly. YouTube does not like automated channels and makes changes to the upload process often. In case the upload script has stopped working, mention it in the forum thread of this ebook. This guide took a lot of testing, scripting, and general research to compile, please enjoy.

Chapter 1

CREATING YOUR CHANNEL

When it comes to short vertical videos YouTube does not care about the age of the account as long as you can verify the phone number and email associated with the account to unlock extra features such as community posts and links in the description. However, YouTube tries to copy TikTok and push the content to the viewers from the country of origin. This means that if you create a channel and upload a short video from Ireland, over 90% of the viewers of the video will be from Ireland and YouTube will try to push it locally as much as possible. Luckily this is not TikTok and can be overcome fairly easily. The recommended method throughout this guide is a dedicated VPS. My personal, fully automated shorts channel is running on a shared VPS with 1 CPU core and 2GB of ram. It takes a lot of time to render, but because it's automated, the speed of it does not matter to me. Virtual servers with such specs can be had from as low as $5/mo. If you are not willing to make any investments without making money first, you can easily run everything on your PC.

Chapter 2

GETTING THE CONTENT

There are a million ways to get content for YouTube shorts, but in this guide, I will show examples with Cartoons and Reddit stories. Both of the example ideas can get a lot of daily traffic, but these might not be ideal niches for monetization. You can rather simply adapt the methods demonstrated below to fit your own ideas and niches.

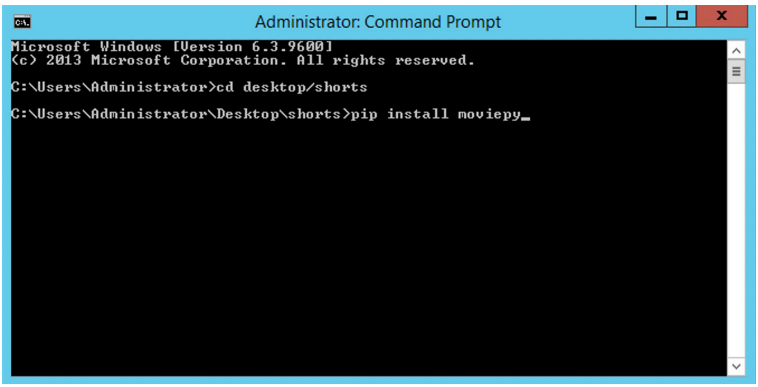
Example 1: Short Reddit horror stories
Example 2: Random funny cartoon clips

BEGINNING THE AUTOMATION

Step 1. You will need **Python** to run the automation. As of the release of this guide, the most stable version for our purposes will be **3.9.16** (not the latest one)

Step 2. Installing required libraries. To make things easier we will install some Python libraries that will help us with the automation process. To install a library just open the terminal and type *pip install library-name*.

Create a new folder for your project, type *cd* and drag&drop the folder on top of the terminal window. Then we can start installing libraries. It should look like this:



Here is the list of all the libraries we will need:

- **MoviePy**
- **Requests**
- **Naked**
- **NLTK**
- **TTS**
- **BS4**

Other requirements:

- **eSpeak TTS module for Windows**
- **Some sort of text editor of your choice**
- **FFMPEG library for Windows**
- **ImageMagic editor software**

Step 3. We will need some content to work with and the first example is going to include Reddit. Not many people know of this feature Reddit has. If you go to any subreddit and type /random in the URL bar, you get a random thread from that subreddit. This means that the system to get content is already in place, we just need to 'scrape it off'. I

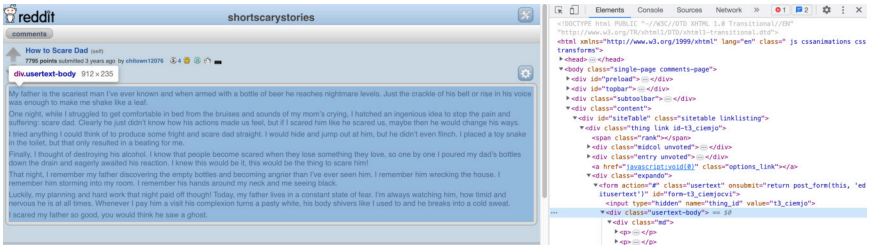
personally like horror stories, so let's try to create some short horror clips using stories from **/r/shortscarystories**.

So we found the content we want, but we don't want to copy-paste it each time ourselves. Let's write a short script that does it for us. As of writing this guide, Reddit still has their old mobile website active, which is super convenient for scraping, as it is simple and loads quickly. We can access it by typing *i.* In front of the reddit.com.

The main goal of this script:

1. Navigate to https://i.reddit.com/r/shortscarystories/random
2. Find the element of the body text
3. Extract the story text from the element
4. Save the story to file

We can easily find the needed element and their classes using inspect tool in any browser. Here is how it looks:



As you can see the main body text of the post just sits between some div tags with a class named '*usertext-body*'. All we have to do is tell our bot to navigate to the webpage, find the element with the same class, and get the text. For this task we will be using ***Requests*** library to access the site and ***BS4*** to interpret the site's HTML. Just like any popular website, Reddit does not like bots crawling their page, but luckily they are not that strict about it as long as you aren't

causing too much trouble. Let's use a random user-agent just in case. If you want to scrape data in greater amounts, you can do so by using their API but for this tutorial, we will keep it simple. Keep in mind that the green lines with hashtags are Python comments mean for the reader to better understand the context. Here is how the code looks:

```python
#Import the required libraries
import requests
from bs4 import BeautifulSoup
#Link to the subreddit of your choice
url = 'https://i.reddit.com/r/shortscarystories/random'
#Set a random useragent to avoid suspicion
headers = {'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/58.0.3029.110 Safari/537.3'}
#Go the the url and get the sites data, allow redirects to get random story
response = requests.get(url, headers=headers, allow_redirects=True)
#Use BS4 (BeautifulSoup4 HTML library) to read the data
soup = BeautifulSoup(response.text, 'html.parser')
# Get the main body text of the post
main_text = soup.find('div', {'class': 'usertext-body'}).text.strip()
# Write the title and main text to a file
with open('video_script.txt', 'w') as f:
    f.write(main_text + '\n')
```

It should be saved with *.py* file extension and can be tested by opening the terminal and typing *py yourscript.py* The script should create a new *.txt* file in the same directory with content similar to this:

```
My father is the scariest man I've ever known and when armed with a bottle of beer he reaches nightmare levels. Just the crackle of his belt or rise in his voice was enough to make me shake like a leaf.
One night, while I struggled to get comfortable in bed from the bruises and sounds of my mom's crying, I hatched an ingenious idea to stop the pain and suffering: scare dad. Clearly he just didn't know how his actions made us feel, but if I scared him like he scared us, maybe then he would change his ways.
I tried anything I could think of to produce some fright and scare dad straight. I would hide and jump out at him, but he didn't even flinch. I placed a toy snake in the toilet, but that only resulted in a beating for me.
Finally, I thought of destroying his alcohol. I know that people become scared when they lose something they love, so one by one I poured my dad's bottles down the drain and eagerly awaited his reaction. I knew this would be it, this would be the thing to scare him!
That night, I remember my father discovering the empty bottles and becoming angrier than I've ever seen him. I remember him wrecking the house. I remember him storming into my room. I remember his hands around my neck and me seeing black.
Luckily, my planning and hard work that night paid off though! Today, my father lives in a constant state of fear. I'm always watching him, how timid and nervous he is at all times. Whenever I pay him a visit his complexion turns a pasty white, his body shivers like I used to and he breaks into a cold sweat.
I scared my father so good, you would think he saw a ghost.
```

Not the horror I was expecting, but ok

Step 3. We have the content, let's make a video! You could do it manually, but that's no fun, so let's write another python script that turns this into a video. This video will consist of:

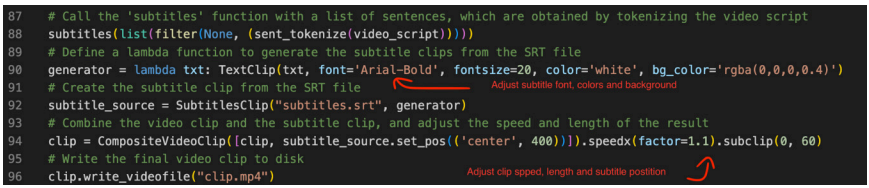
- Stock background footage
- Overlay story subtitles
- Text to speech VoiceOver
- Background music

For this demo, I will be using this background video, and this background music. Now that we have everything we need, let's get started. The goal of this script is to crop the stock background footage to a 9:16 aspect ratio, use TTS to turn the Reddit story into audio, chop it into separate lines of text that would fit on the screen, use TTS to turn these lines into audio and overlay it with some background ambiance. Sounds like a good recipe? The code for this task is quite long, but don't worry, it's not that scary when you read it. For convenience sake, I divided it into two parts:

```
# Import the necessary libraries and modules
import nltk  # for natural language processing tasks
import datetime  # for date and time manipulation
from TTS.api import TTS  # text to speech package
from moviepy.editor import *  # video editing package
from nltk.tokenize import sent_tokenize  # sentence tokenization from natural language
from moviepy.video.tools.subtitles import SubtitlesClip # tools for reading and displaying subtitles
# Download the 'punkt' module used for sentence tokenization if it has not already been downloaded
nltk.download('punkt')

# Select the model for text-to-speech conversion
model_name = TTS.list_models()[12] <-- In my opinion the best English voices of this library are: 6, 9, 12 & 20
# Create an instance of the selected text-to-speech model
tts = TTS(model_name)
# Open the text file containing the video script, and read the contents
video_script = open('video_script.txt', 'r').read()
# Convert the video script to an audio file using the selected text-to-speech model
tts.tts_to_file(text=video_script, file_path="voiceover.wav")
# Load the newly created audio file, and adjust the volume of the background music
new_audioclip = CompositeAudioClip([
    AudioFileClip("voiceover.wav"),
    AudioFileClip('background_music.mp3').volumex(0.2) <-- Adjust depending on background music
])
# Load the video file that will be used as the background of the final clip
video = VideoFileClip('background_video.mp4')
# Determine the dimensions of the video, and calculate the desired width based on the aspect ratio of 16:9
width, height = video.size
new_width = int(height * 9/16) <-- This makes any video vertical for YT shorts
# Crop the video to the desired width, centered horizontally
clip = video.crop(x1=(width - new_width) / 2, x2=(width - new_width) / 2 + new_width)
# Set the audio of the cropped video to the adjusted background music and voiceover audio
clip.audio = new_audioclip
```

At the beginning of this code, we define the necessary libraries and download the NLTK module that helps to divide our script into smaller sentences for subtitles to fit the screen. Then we select the desired TTS voice. When trying this code for the first time TTS library will download the selected voice. This particular library has almost 100 voices to choose from and about 10% of them are *very* good in my opinion. There are also quite a few languages besides English. Here is the full list. Just change the *model_name* number to choose the voice you want. The second part of this code is fairly straight forwards, we just declare what files we will be using to create this video and then cut the background video to fit the vertical format. Let's move on to the second part of the code:



First, we begin by creating the subtitles, then set their styling, overlay them over the main video, and fine-tune the results by changing the clip speed and length. Some of the stories in this subreddit will not fit into a 60 second short video. There are multiple ways to solve it (find shorter stories, make multiple parts, upload the full story as a regular video, etc.) In this example, videos will be cut at exactly 60-seconds to qualify as shorts. As you might have noticed there is no code to generate subtitles. In line 88 we call a function named *subtitles*, but I haven't included it in this screenshot, because it's large and complicated. Think of it like a car - you don't need to know how the engine

works to drive it. I tried my best to comment every line to make it as easy to follow as possible, but the only thing that might need adjusting is the max word count per line and subtitle speed. Here is how the function looks:

```
# Define a function to create subtitles for the video
def subtitles(sentences):
    # Initialize an empty list to store the SRT file contents
    srt_lines = []
    # Initialize the start and end time to zero
    start = datetime.timedelta(seconds=0)
    end = datetime.timedelta(seconds=0)
    # Initialize a counter to keep track of subtitle numbers
    counter = 1
    # Loop over each sentence in the list of sentences passed to the function
    for sentence in sentences:
        # Split the sentence into words
        words = sentence.split()
        # Calculate the number of lines needed for this sentence (assuming each line has 4 words)
        num_lines = len(words) // 4 + 1
        # Loop over each line of the sentence
        for j in range(num_lines):
            # Get the words for this line
            line_words = words[j * 4: (j + 1) * 4]
            # Join the words into a single string to form the line
            line = ' '.join(line_words)
            # Calculate the end time for this line based on the length of the line
            end += datetime.timedelta(seconds=len(line_words) * 0.35)  <-- Adjust subtitle speed
            # Check if the line is not empty
            if line:
                # Format the start and end times as strings in the SRT format
                start_str = '{:02d}:{:02d}:{:02d},{:03d}'.format(
                    start.seconds // 3600,
                    (start.seconds // 60) % 60,
                    start.seconds % 60,
                    start.microseconds // 1000
                )
                end_str = '{:02d}:{:02d}:{:02d},{:03d}'.format(
                    end.seconds // 3600,
                    (end.seconds // 60) % 60,
                    end.seconds % 60,
                    end.microseconds // 1000
                )
                # Add the subtitle number, start and end times, and line to the SRT list
                srt_lines.append(str(counter))
                srt_lines.append(start_str + ' --> ' + end_str)
                srt_lines.append(line)
                srt_lines.append('')
                # Increment the subtitle counter
                counter += 1
            # Update the start time for the next line
            start = end
    # Join the lines of the SRT file into a single string
    srt_file = '\n'.join(srt_lines)
    # Write the SRT file to disk
    with open("subtitles.srt", "w") as f:
        f.write(srt_file)
```

I marked the number that changes the subtitle speed (less = faster), but to change the max word count per 1 subtitle line you will need to change all the 4s to a desired number. The current setting is max 4 words per subtitle line as with the current font it's difficult to fit more in a vertical video.

This is only meant to introduce the code and make it seem more friendly for beginners, the complete and ready-to-be copied scripts will be shared at the end of this guide.

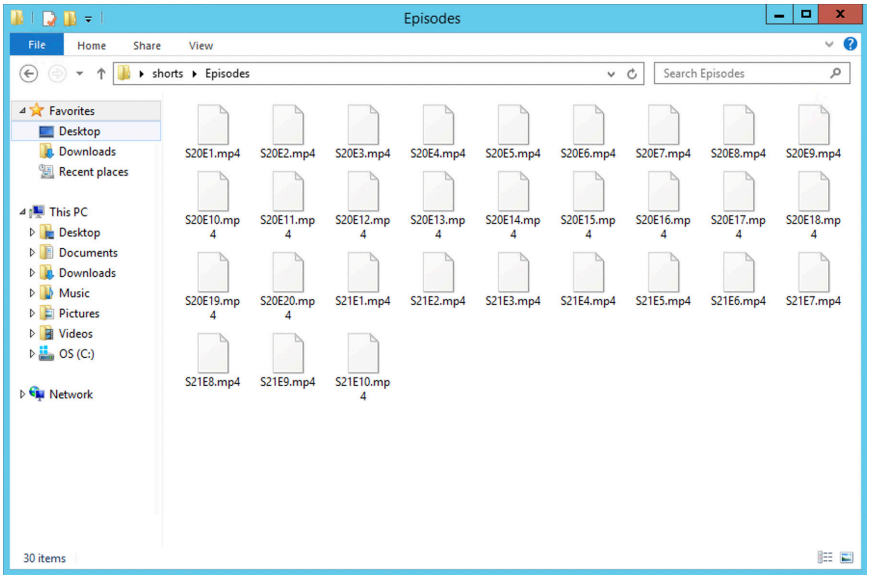
End of example 1.

Now we have 2 script that can be easily joined to generate endless short horror video stories. I do not think this is a production ready copy-paste example, but I am sure with some ideas of your own and slight tune of the code you can auto generate amazing content that will get thousands of views per upload. Curious to see an example that was generated by the script above? **Here it is!**

You might be thinking, but how the content is endless if you are just using one background video and same music over and over again? That's right, let's continue on to **example 2** to learn about 'content folders' and how to randomize elements to produce unique videos each time.

Example 2

This is the easiest way to get traffic from YouTube shorts. I tried it multiple times and the views were overwhelming each time. It might not be the ideal niche for monetization, but traffic is traffic and you can figure out how to get some money out of it regardless. In this example, I will show you a script that cuts random clips from popular tv shows or cartoons, makes it a bit more difficult to detect for the YouTube algorithm (even though shorts allow some copyrighted content), and finally add some text that encourages viewers to check the comment section. There is not much introduction necessary as the code is quite short and simple. However, you will need a folder with .mp4 files of your favorite tv show episodes to get started. Here is a folder I have of some cartoon episodes:



Now that you have a folder with some content ready, let's look at the code that turns it into shorts:

```python
import os  # importing os module for interacting with operating system
import random  # importing random module for generating random values
from moviepy.editor import *  # importing necessary classes from moviepy module
from moviepy.video.VideoClip import TextClip

# creating a list of .mp4 files in the 'Episodes' directory using list comprehension
mp4_files = [file for file in os.listdir('Episodes') if file.endswith('.mp4')]
# randomly choosing a file from the list
random_file = random.choice(mp4_files)
# creating the full path of the chosen video file
video_file = os.path.join('Episodes', random_file)
# loading the video file using VideoFileClip() class
video = VideoFileClip(video_file)
# getting the duration of the video
duration = video.duration
# choosing a random start time between 30 seconds and 60 seconds before the end of the video
start = random.uniform(30, max(30, duration - 60))
# choosing a random length between 20 seconds and 40 seconds
lenght = random.randint(20, 40)
# extracting the clip from the video using the chosen start time and length
clip = video.subclip(start, start + lenght)
# getting the width and height of the clip
width, height = clip.size
# calculating the aspect ratio of the clip
aspect_ratio = width / height
```

Don't be intimidated by the size of this script, as 60% of it is just comments.

```
# calculating the new width of the clip with a 16:9 aspect ratio
new_width = int(height * 9/16)
# calculating the left margin to center the clip horizontally
left_margin = (width - new_width) / 2
# cropping the clip to the new width and centering it horizontally
clip = clip.crop(x1=left_margin, x2=left_margin + new_width)
# increasing the speed of the clip by 10%
clip = clip.speedx(factor=1.1)
# flipping the clip horizontally
clip = clip.fx(vfx.mirror_x)
# creating a TextClip object with the desired text and properties
text = "Surprise in comments\nEnter & WIN!"
txt_clip = TextClip(text, fontsize=15, color='white', font='Arial-Bold')
# setting the position of the text clip to be centered near the bottom of the screen
txt_clip = txt_clip.set_position(('center', 0.8), relative=True)
# creating a CompositeVideoClip object by combining the clip and text clip
final_clip = CompositeVideoClip([clip, txt_clip])
# setting the duration of the final clip to be the same as the cropped clip
final_clip.duration = clip.duration
# writing the final clip to a file named 'clip.mp4'
final_clip.write_videofile("clip.mp4")
```

Just like the previous script we start by importing the needed libraries and choosing the files. Unlike the previous example, we do not choose a single file, but rather a whole folder and pick an episode randomly. You can use this method for all the content from the previous example and generate videos with random backgrounds, music, and other elements picked by the script each time. It is also possible to randomly scrape this content from other online media sources. This eliminates the need to switch the content manually each time. You can also write a simple addition script that prevents duplicates as that might happen sometimes. Getting back to the code, we take a random .mp4 file and cut a short clip from it, excluding the intro and end credits. Then we mix it up by mirroring the clip and slightly speeding it up. After the clip is done we overlay some text for our monetization needs and render the final product. It's barely noticeable that the video is mirrored, cut, and sped up, most short viewers will enjoy it regardless. Want to see a video generated by this code? **Here it is!**

Preparing for takeoff.

Just like everything you upload on YouTube, these videos will need titles, descriptions, and tags to get those views.

```
1  pos1 = ['Deleted scene', 'Cool new episode', 'Funny moment'] # Some ideas for the title
2  pos2 = ['Try not to laugh', 'Watch this video', 'Unexpected'] # Some ideas for the description
3  with open("count.txt", 'r') as file: # Count the video amount and include it in the title
4      counter = file.read() # Read the current counter number
5  title = f"Cartoon {random.choice(pos1)} #{counter} ({os.path.splitext(random_file)[0]})\n" #Generate a random title
6  description = f"""{random.choice(pos2)} from Family Guy {os.path.splitext(random_file)[0]} | Enter a contest: your_link_here
7  #shorts #cartoon #funny #tags #youtube #guides
8  """ # Generate a random description for the video
9  tag_list = ["cartoons", "funny moments", "amazing episodes", "great tags", "blackhatworld"]
10 tags = random.choice(tag_list)+','+random.choice(tag_list)+','+random.choice(tag_list) # random 3 tags (add as many as you wish)
11 with open("upload_text.txt", 'w') as file: #create a new file
12     file.write(title+description+tags) #write everything to file
```

We sure don't have time to write it all ourselves every time, so here is some code we can add to generate dynamic text for our uploads. I would recommend adding as many options for the title, description, and tags as possible, so each generation would be somewhat unique and less repetitive. Usually, you would want to max out the tags as well for the best ranking. As you can see the majority of the code is just arrays filled with options, which are randomly selected and saved to file in the end. In line 5 we can see that the title will always start with the word 'Cartoon' followed by a randomly chosen option from variable *pos1*, then we have a # with a counter variable that returns the number of our video, and finally the file name between the brackets. Assuming the file is named "Simpsons S12", a possible outcome could be "Cartoon Deleted Scene #2 (Simpsons S12)". The '\n' command is added for a new line. The same thing happens with the descriptions, except the triple quotes enclosing the code mean that it is in multi-line mode and no '\n' commands are needed to change lines. Adjust this as per your needs to build the desired dynamic text generator. As you can see line 10 is a very basic attempt to join 3 tags together, but usually, we want around 10

relevant tags to rank our video as well as possible. This can

```python
import random

tag_list = ["cartoons", "funny moments", "amazing episodes", "great tags", "blackhatworld"]

tags = []  # Initialize an empty list to store the generated tags

for i in range(10):
    tag = random.choice(tag_list)  # Generate a random tag from the tag_list
    tags.append(tag)  # Add the generated tag to the tags list

tags_string = ', '.join(tags)  # Join the tags list into a string with commas and spaces in between
```

be achieved with a simple loop like this:

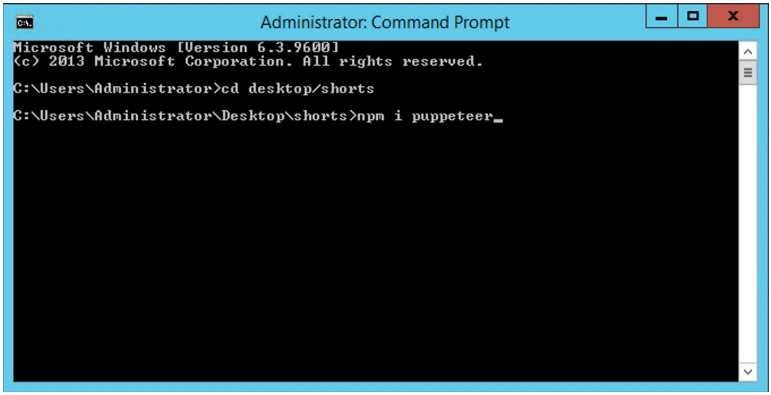
You might notice that the last line of the description also includes some tags, why is that? As previously mentioned YouTube is trying to be TikTok and started to prioritize tags that are in title or description instead of the ones buried deeply in the metadata. This means that it is currently beneficial to include tags not only in the dedicated spot, but sprinkle some in the video titles or descriptions for that ultimate ranking boost. Don't forget that #shorts!

After a bit of practice and trying out both of these examples, you should be able to make your script that chooses the content and turns it into a short video suitable for YouTube shorts. The quality of that video will mostly depend on the materials chosen, but overall it's currently very easy to get traffic, even if your videos aren't the best. Some people do the Reddit stories with Minecraft parkour backgrounds and **TikTok TTS** voiceovers. As we all know those videos are terrible, but get millions of views anyway. The sky is the limit, use your imagination and automatically generate videos that can be uploaded multiple times a day to get thousands of clicks.

Chapter 3

AUTOMATING THE UPLOADS

So now we have a video made by a bot, but can a bot upload it too? Sure can! Let's create a script that uploads our videos to complete this automation challenge. For this method, we will be using **NodeJS & Puppeteer** instead of **Python & Selenium**, because Selenium uses a web driver that can easily be detected and is not welcome by most websites, including YouTube. Puppeteer is not fool-proof either, but I have been using this method for months and it's working flawlessly. Keep in mind that it is officially **not** allowed to automate the upload process on YouTube, so I am obligated to say that I am not responsible for any issues caused. To begin, download NodeJS and install the required libraries similarly to before. To begin this project, we will need the Puppeteer library, which you can get just like this:



Once that is installed, let's grab delay-random library as well. It's not necessary, but very convenient and does exactly what it says. When the node packages are in place and you can see *node_modules* folder present, we can create a new file named *uploader.js* and start working on the code. Now please be warned that this is some messy, low-level code, but due to how often YouTube changes the upload process, it makes it easier to implement any changes:

```
// Import required packages
const fs = require('fs'); // For reading files
const puppeteer = require('puppeteer-core'); // For browser automation
const url = 'https://studio.youtube.com/'; // URL to navigate to
const delayRandom = require('delay-random'); // For adding random delays

// Read the upload text that was created previously
fs.readFile('upload_text.txt', 'utf8', function(err, data) {
    if (err) throw err; // Show if something is wrong with the file
    lines = data.split('\n'); // Split the file into individual lines
});

// Launch Puppeteer browser instance with a local cache storage
puppeteer.launch({
  userDataDir: './uploader_cache', // The location where the cache will be stored
  executablePath: require('puppeteer').executablePath(), // Path to the executable of a specific version of Chrome (installed by Puppeteer)
  headless: false, // Run the browser in GUI mode
  args: ['--no-sandbox'] // Arguments to pass to the browser
})
.then(async browser => {
    const page = await browser.newPage(); // Open a new page
    await page.setCacheEnabled(true); // Enable the cache to save the sessions
    await page.setViewport({ width: 1280, height: 720 }); // Set the size for the browser window
    // Read the cookies from a JSON file
    const cookies = JSON.parse(fs.readFileSync('cookies.json', 'utf8'));
    // Add the cookies to the page
    for (const cookie of cookies) {
    await page.setCookie(cookie);
    }
    await page.goto(url); // Go to the YouTube Studio website
    await delayRandom(14000, 26000); // Wait for the website to load (adjust based on your internet speed)
    // From this point on, the script interacts with the website by clicking buttons and entering text
    await page.click('#create-icon'); // Click on the 'Create' button
    await delayRandom(1000, 2000); // Wait for the next step to load
    await page.click('#text-item-0'); // Click on the 'Text' option
    await delayRandom(1000, 2000); // Wait for the next step to load
    await page.click('#select-files-button'); // Click on the 'Select files' button
    await delayRandom(1000, 2000); // Wait for the next step to load
    const elementHandle = await page.$("input[type=file]"); // Get the file input element
    await elementHandle.uploadFile('clip.mp4'); // Upload the video file (make sure everything is in the same folder)
    await delayRandom(8000, 10000); // Wait for the file to upload (adjust based on your file size)
    await page.keyboard.type(lines[0], {delay: 150}); // Enter the title of the video
    await delayRandom(1000, 2000); // Wait for the next step to load
    await page.keyboard.press('Tab'); // Move to the next input field
    await delayRandom(500, 600); // Wait for the next step to load
    await page.keyboard.press('Tab'); // Move to the next input field
    await delayRandom(1000, 2000); // Wait for the next step to load
    await page.focus('#description-container'); //focus on the description container
    await delayRandom(1000, 2000); //wait for a random amount of time
    await page.keyboard.type(lines[1], {delay: 150}); //type the first line of the description with a delay between keystrokes
    await delayRandom(1000, 2000); //wait for a random amount of time
    await page.keyboard.type(lines[2], {delay: 150}); //type the second line of the description with a delay between keystrokes
    await delayRandom(1000, 2000); //wait for a random amount of time
    await page.click('#audience > ytkc-made-for-kids-select > div.made-for-kids-rating-container.style-scope.ytkc-made-for-kids-select > tp-yt-paper-radio-group > tp-yt-paper-radio-button:nth-child(2)'); //click the "No, it's not made for kids" radio button
    await delayRandom(1000, 2000); //wait for a random amount of time
    await page.keyboard.press('Tab'); //press the Tab key
    await delayRandom(1000, 2000); //wait for a random amount of time
    await page.keyboard.press('Tab'); //press the Tab key again
    await delayRandom(1000, 2000); //wait for a random amount of time
    await page.keyboard.press('Enter'); //press the Enter key
    await delayRandom(1000, 2000); //wait for a random amount of time
    for (let i = 0; i < 8; i++) { //loop 8 times
        await page.keyboard.press('Tab'); //press the Tab key
        await delayRandom(300, 600); //wait for a random amount of time
    }
    await delayRandom(1000, 2000); //wait for a random amount of time
```

```
    await page.keyboard.type(lines[3], {delay: 150}); //type the third line of the description with a delay between keystrokes
    await delayRandom(1000, 2000); //wait for a random amount of time
    await page.click('#next-button'); //click the "Next" button
    await delayRandom(1000, 2000); //wait for a random amount of time
    await page.click('#next-button'); //click the "Next" button again
    await delayRandom(1000, 2000); //wait for a random amount of time
    await page.click('#next-button'); //click the "Next" button a third time
    await delayRandom(1000, 2000); //wait for a random amount of time
    await page.click('#offRadio'); //click the "Not made for kids" radio button
    await delayRandom(1000, 2000); //wait for a random amount of time
    await page.keyboard.press('Tab'); //press the Tab key
    await delayRandom(1000, 2000); //wait for a random amount of time
    await page.keyboard.press('ArrowUp'); //press the Up Arrow key
    await delayRandom(15000, 20000); //wait for a random amount of time
    await page.click('#done-button');
    await delayRandom(15000, 20000);
    // At this point, the video should be uploaded. The following code navigates to the newly uploaded video's page and leaves a comment.
    // Get the URL of the newly uploaded video
    const element = await page.$('#share-url');
    const share = await page.evaluate(el => el.innerHTML, element);
    await page.goto(share);
    await delayRandom(10000, 15000);
    // Click the "Comments" button to expand the comments section
    await page.click('#comments-button');
    await delayRandom(5000, 10000);
    // Press the "Tab" key several times to navigate to the comment text box
    await page.keyboard.press('Tab');
    await delayRandom(1000, 2000);
    await page.keyboard.press('Tab');
    await delayRandom(1000, 2000);
    await page.keyboard.press('Tab');
    await delayRandom(1000, 2000);
    // Type the comment text into the text box
    await page.keyboard.type('Special prize for you: cpalinkhere', {delay: 150});
    await delayRandom(1000, 2000);
    // Click the "Submit" button to post the comment
    await page.click('#submit-button');
    await delayRandom(15000, 20000);
    // Print a message to indicate that the upload and comment posting are complete, and exit the script
    console.log('Upload completed')
    process.exit();
})
.catch(function(error) {
    console.error(error);
    //process.exit();
});
```

As you can see it just goes through the upload process as a human would and uses keyboard navigation for the elements it cannot reach by mouse. At the beginning of the code, the *upload_text.js* is read and stored in an array line by line. Assuming that line one is the title, line 2 is the description, and the last line is the tags. The text is accessed and typed by this command:

await page.keyboard.type(lines[0], {delay: 100});
Line 0 is the first line of the file, the delay simulates human typing.

Before we can try this code out, you will need the *upload_text.txt* file we created using the previous script as well as the cookies from your YouTube account, to access it. Open your YouTube studio as you usually do and install this extension then export all the cookies from the site. This should leave you with a file called *cookies.json*. Now we can

take the upload bot for a spin. Open the terminal and type *node uploader.js* to launch the script and begin the upload process.

There are many ways to improve this code as it is as basic as it gets, so that beginner readers would not have too much trouble understanding what is going on with it. I would not call this code very elegant either, it is just forcing its way through the upload page, but I do not think it matters as long as it achieves the desired action. Feel free to improve and make this bot more sophisticated.

As you might have noticed I am not using a headless browser, meaning that you will be able to see a browser window opening and the script controlling it. Many people would prefer using the headless mode to hide the browser and let the action happen in the background, but unfortunately, YouTube easily detects headless attempts and closes them down. If you have a method to upload in headless mode please share, but for general automation purposes, this works just fine.

If you do not wish to add comments to all your uploaded videos, you can remove everything from '#share-url' to 'upload completed' and have a regular uploader script.

If you wish to change the video category or any other upload options, add the necessary code after line 68. This is a part of the code where the tags are entered and most menus for category or language changes are accessible.

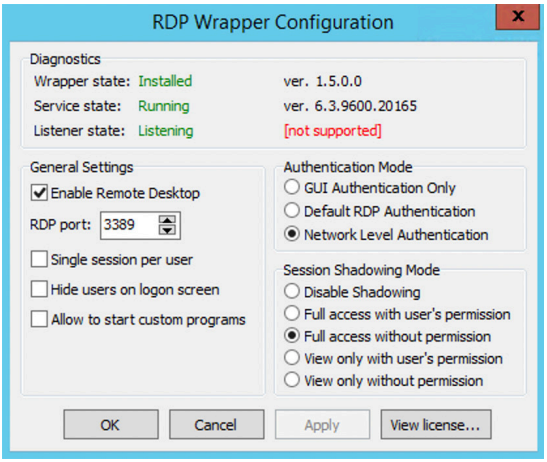
Chapter 5

FINAL STEPS

Setting everything up

Wonder how to set everything up on a VPS to get passive traffic going without any user input just as promised? Let's run through the final steps to fit all the pieces together and reach that satisfying total automation. This will allow you to keep the traffic coming even when you sleep, isn't it wonderful?

Step 1. Due to the upload script not working in a headless mode we will need to trick Windows into thinking that someone is logged in to prevent it from closing the GUI. This can be easily done with an old utility called 'RDP Wrapper' I am using an old Windows Server VPS, so I cannot guarantee that it still works on the latest Windows 11 images, but most VPS providers allow you to choose an older version for compatibility purposes anyway. Install the software and try opening the configuration .exe file. This piece of software might trigger some antivirus programs, but it is a false positive with no harm. Set your configurations to be like this:



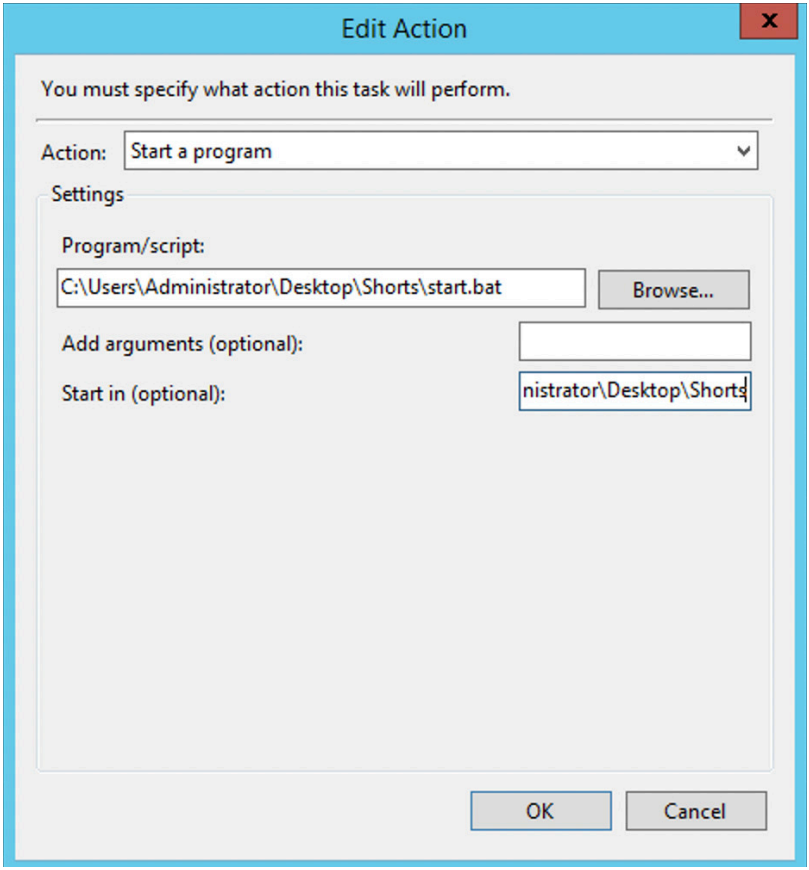
If everything was done correctly you should be able to open the RDP window and connect to your own desktop ‘remotely’ just like this:

By keeping this connection active and working on the ‘remote’ computer you trick Windows into thinking that there is always someone logged in. Essentially, it just creates an endless loop where one user is dependent on another user.

Step 2. Let’s pretend we are going with the cartoon clip niche and we want to upload daily clips of random cartoons. To make this happen we need to trigger our script every day at a selected time. This can be easily set up using the built-in Windows Task Scheduler. To be able to launch a python script from this application, we will need to create a *helper* script. Open a new text file and write this batch command in a single line:

```
"C:\Users\Administrator\AppData\Local\Programs\Python\Python39\python.exe"
“C:\Users\Administrator\Desktop\folder-name\yourscript.py”
```

Adapt them as necessary to work on your machine.
Now save this file with a *.bat* extension, open the Task Scheduler, and select ‘Create basic task’. Name it however you want and select how often you want the videos to be uploaded. When you reach a ‘start program’ page, browse and select the *.bat* file we have created previously. Now copy the path of the .bat file and paste it into the ‘Start in’ field without typing in the file itself. It should look like this:



Once that is set, save the task and adjust the timing as needed, there are plenty more detailed tutorial on YouTube as this guide is not about the Windows Task Scheduler.

Step 3. Now we have a python script that will get triggered once (or more times) a day automatically and generate video on it's own. But our upload script is on a separate file in a different language, how do we join them? To do this, we tell the python script to launch the uploader script after the video is ready, by using this simple command:

execute_js('uploader.js')

It's really that simple! However, for that command to be accessible you do need to include the *Naked* module first.

from Naked.toolshed.shell import execute_js, muterun_js

Now every time the video generator script is launched, it will prepare a new video and automatically call the uploader to upload it.

Conclusion.

Make sure you have read through the code and are familiarized with the commands. This might be a bit tricky to orchestrate for the first time, so here is a flow chart that should help you understand better:

Task Manager set to launch *start.bat* at 1 am ->
Time is 1:00 -> *start.bat* is active ->
The batch command triggers *cartoon_short.py* ->
Video is generated and saved as clip.mp4 ->
Video title & description generated and saved as desc.txt ->
Python is calling NodeJS uploader.js script ->
uploader.js looks for cookies.json & desc.txt ->
uploader.js launches Chrome and starts uploading ->
uploader.js finishes the upload and adds a comment ->
uploader.js closes -> *cartoon_short.js* closes -> task finished.

I am sure that there are many different ways to do this, but the basics are here for you to grab and apply. This should not take longer than an hour to set up in total and the traffic you will get passively should be truly impressive.

Chapter 4

SHARING IDEAS

Let's begin this chapter by talking about the frog 🐸 in the room - Coqui TTS. It's the python TTS library we have used in the first example. This library has plenty of options and settings to play around with to get that perfectly realistic human voice out of a single file on your computer. I agree that the voice in the example is not perfect, but if you take a moment to read through the library's documentation and learn how to adjust and fine-tune the details as well as import custom voice packages from other creators and alter word pronunciation or use multiple speakers at once. I would also like to remind everyone that there are more languages than just English and some content might be even easier to rank and get views in other languages. If you have an Nvidia GPU you can also use your CUDA cores to speed up the TTS generation and render videos in seconds.

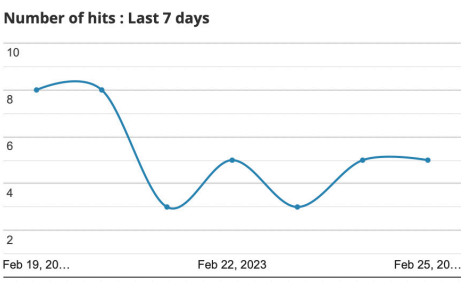
Now you probably noticed that the monetization examples in this guide are not the best. However, this guide is about teaching how to get large amounts of free traffic. It is up to the individual reader how to monetize or utilize that traffic. That said, I will share one beaten-to-death idea, that still somehow works. Game hacks. With a tiny bit of brain power, you can easily adapt the scripts from this

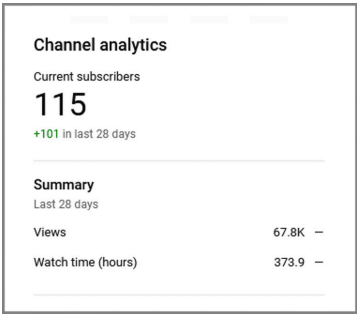
guide to generate endless short game hack videos with download (CPA) links in the comments. There are plenty of subreddits sharing modded APK files and there is already a way for you to get those posts. All left to do is add some extra elements to make the videos more appealing to a younger audience and you are making money from the automated YouTube Shorts channel! I tried it myself and it works, however, I would only recommend this niche if you are desperate, because the competition is massive.

I bet there are at least a million niches that could be used in conjunction with this automated shorts method. Take your time to test the scripts out, get familiar with each line of code and adapt it to any niche to make that money.

How much traffic can you get?

Here is a screenshot from a channel I created 1 month ago. All the content uploaded is just movie clips using the method from example #2. I only logged in to this account a few times to check the stats. All the traffic was made completely passively with no input from me. I did try to include a link in one video but have not made any effort to promote it. You can see the clicks in the first graph. This channel is no longer active as I am working on different niches right now, but it is still getting decent daily views.





I truly think that this shows how much it can be scaled. Imagine creating 10 channels that are completely automated and easily getting 1000 clicks to your CPA offer every day. This is possible if you take action and invest the time needed to set everything up in a sensible way. Don't skim out on the content either. It does not mean that automated videos have to suck. Some of them can truly be entertaining and valuable to the short community.

There is also always a possibility to get monetized by YouTube directly. All you have to do is fine-polish that robotic voice to sound more like a human, spend more time cleaning up the raw content to fit the video format better, and reach a level of production that might be deemed acceptable by YouTube gods.

Let's not forget TikTok and the massive user-base there. I do now have a reliable method to share on how to upload content automatically to TikTok, but I am sure that some of the scripts from this guide can be adapted to automate traffic from any platform.

If you come this far into this guide, thank you for reading and I hope it was a valuable piece of information. You will find all the code from previous examples below. Feel free to copy and test it for yourself. If you have any issues check out the troubleshooting page or contact me directly on BHW. Now go make that traffic start flowing!

CODE DUMP

Individual pieces of code:
Scraping Reddit for content -> **Pastebin**
Generating videos from stories -> **Pastebin**
Cutting clips from tv episodes-> **Pastebin**

Complete scripts:
Reddit to video -> **Pastebin**
Cartoon clips -> **Pastebin**

YouTube uploader:
Video upload -> **Pastebin**

TROUBLESHOOTING

Q: My code doesn't work, what do I do?
A: Paste it in chatGPT and ask what is wrong with it.

Q: My code still doesn't work, what do I do?
A: Contact me on BHW and I will try to help you

Q: It generates the video, but doesn't upload
A: Make sure all the required files are in the same folder

Q: I installed ImageMagick, but it still shows it's missing
A: Select all the checkboxes when installing

Q: I installed FFMPEG, but it still shows it's missing
A: Try a different, older version, like this one

Q: How do I install FFMPEG, it's just a bunch of files?
A: Like this

Q: I installed *thing* but it's not working
A: Have you added it to PATH? Have you tried restarting?

Q: Can't upload, says browser not supported
A: The cookies are corrupted or missing

Q: The code provided doesn't work (moviepy error)
A: Wrong version of Python or missing dependencies