

Forum:[Hack Forums](#)

Thread: [Python versus Batch for General Utility Scripting](#)

Sajuuk - July 14, 2020, 11:42 pm

I know this section is pretty well dead, with even the more active members remaining almost exclusively on Discord. However because I don't trust discord enough to actually use it, I'll post this here.

My question is would it be advisable to consider moving to Python from Batch for general utility scripting (in the sense of automating simple command-line mathematical or file operations). The reason I'm asking the opinion of others, and more over posting this here rather than in the Python section, is because obviously there will be more people well-experienced with Batch who can give a bit more detailed insight as to why or why not.

This topic came to mind a few weeks ago when I decided to completely reformat all of my music library from the same old 41k MP3 I've been using for years. I began considering it after reading about the witchcraft benefits of the OPUS format. (Essentially you retain a considerable amount of quality even at very low bitrates.) Now, I'm not all too familiar with the technical aspects of how Batch goes. I've enough experience with it over the years to the point where I can look up something I need to do and not have much if any issue instating whatever it may be into my script. However upon beginning the script for automating the entire downloading and conversion process in one script, I came across an issue with tokenizing the file name in order to specify the artist and title individually (for use in metadata).

FFmpeg allows for full access to metadata tagging via command line. That was perfect as I just wanted to run a single script and have everything be done. However the specific issue I had was with splitting the file name and removing the extension. (I've since completed the script in Python and deleted the old Batch script, so I'll do my best to recall exactly what went on.) I used the string manipulation functions in an attempt to split/remove what was needed. Specifically, I just needed to tokenize two elements between a string, that being " - " (a space followed by a hyphen, then an additional space). After experimentation, it appears as if Batch's string manipulation function does not allow for splitting by a string, only by an individual character. Correct me if I've misinformed myself. I read up on Robvanderwoude's archive which I've used for years but I may very well have mistaken or misread something.

At that point, I searched for a Batch Regex script. Turns out, there's JREPL. (If you've some need to Regex string replacement in Batch, it's a fantastic little tool.) I imported it and began using it, however for whatever reason (yes, I used the correct Regex formatting for what I needed) it would not allow for individual tokenizing of each element into separate variables. For example, I ran "Nickelback - Photograph.webm" as the input file, and it split between the " - " like it should. But for whatever reason it included the " - " in the initial token. Therefore the artist was set as "Nickelback - " and the title was (as proper after removing the last four characters with string manip) "Photograph". Perhaps I was a bit lazy and frustrated at the same time, in addition to knowing damn well that it would be so much easier in Python. However because I'm stubborn, I kept at it but to no avail. After two or so hours of tinkering around, I caved and began the same script in Python. Low and behold the Python version (up to this point) has taken only a mere three hours.

Granted, I understand that everyone has their own preferred language and way of doing things in said language. However for the longest time I have been forcing myself to use Batch for

(presentably) such simple operations. Maybe I just went about it in an inefficient manner, I don't know. But Python came so much easier (as I am used to C and C++, which Python offers capability similar to arrays). That's what I'm asking. What do you think? Is Python better overall for general utility programming? It's not system-native, but performance is not of my concern for some generic automation script. I mean, it offers system shell executing from within Python without the need for external scripts like many other languages. Additionally, it has a much wider variety of external libraries. I feel like this is sounding more of a sales pitch for Python, but I don't mean it to come off that way. I'm curious as to what you guys here have to say about the comparisons. I, myself, am almost completely set on moving to just Python with little to no Batch (unless, of course, it's something trivial where Python provides no better solution). Thoughts?

Also, this isn't in the Python section, but I can recreate the incomplete Batch script and additionally post the complete Python script for comparison. Actually... I'll see if I can actually complete the script in just Batch again. I'm a glutton for punishment, I guess.

HFDone - July 15, 2020, 2:06 am

this exactly why powershell came for

Sajuuk - July 15, 2020, 5:31 am

(Jul 15, 2020 02:06 AM)HFDone Wrote: [\[CITE\]](#) this exactly why powershell came for
Here about two years ago I did some experimenting with Powershell in an unrelated project (iirc it was decompression of HOD containers and extraction of the DDS and TIF format textures within). I'm not sure why I stopped using it. Maybe just the formatting or something, I don't know. Almost seems as if it's unnecessary but it has its own strengths over Batch. Now, after further researching this topic, I've completed the all-Batch version of the script. Perhaps my initial thought of the impossibility of an all-Batch script was a brainfart. Here are the two for comparison. I don't think I've optimized them to some extreme extent, so either of them may can be made shorter or more inline (assuming some readability is sacrificed and that the line escape character isn't used to make it a one-liner).

Batch:

```
Code @echo off
setlocal enabledelayedexpansion
set "istream=%~dp0istream"
set "ostream=%~dp0ostream"
set "links=_videolist.txt"
set "iext=.webm"
set "oext=.opus"
set "delim=_ "
youtube-dl.exe --newline -f bestaudio -o "!istream!\%%(uploader)s - %% (title)s.%%(ext)s" -a
"%links%"
for %%a in (!istream!\*!iext!) do (
    set file=%%~na
    call set file=%%file: - Topic - =!delim!%%
    call set file=%%file: - =!delim!%%
    call :tokenize "!file!" "%delim%"
    call set file=%%file:~!delim!= - %%
    ffmpeg.exe -i "%a" -vn -acodec copy -metadata artist="!artist!" -metadata title="!title!"
"!ostream!\!file!!oext!"
)
goto :end
:tokenize
for /f "tokens=1,2 delims=%~2" %%A in ("%~1") do (
```

```

    set artist=%A
    set title=%B
)
goto :eof
:end
endlocal
exit /b
Python:
Code import os
istream = os.getcwd() + "\\istream\\"
ostream = os.getcwd() + "\\ostream\\"
links = "_videolist.txt"
iext = ".webm"
oext = ".opus"
ytdl = "youtube-dl.exe --newline -f bestaudio -o \"\" + istream + "%(uploader)s - %(title)s.%(ext)s\" -a
\"\" + links + "\"\"
os.system(ytdl)
for file in os.listdir(istream):
    if (file.endswith(iext)):
        tmpfn = file
        if tmpfn.find("Topic - ") != -1:
            tmpfn = tmpfn.replace("Topic - ", "")
        else:
            continue
        tokens = tmpfn.split(" - ")
        command = "ffmpeg.exe -i \"\" + istream + file + "\" -vn -acodec copy -metadata artist=\"\" +
tokens[0] + "\" -metadata title=\"\" + tokens[1].replace(iext, \"\") + "\" \"\" + ostream + tmpfn.replace(iext,
oext) + "\"\"
        os.system(command)
    else:
        continue

```

Language-specific differences are as follows: Batch requires the use of a custom subroutine for string tokenization by delimiter. Python uses its own string functions to tokenize. In the Batch script, I've set the custom delimiter to the underscore, as it's uncommon if existent at all in song titles and artist names (at least in the music I listen to). The subroutine tokenizes by underscore and sets the artist/title variables accordingly, whereas Python's string functions allow for direct editing of strings which can be passed as return values by whatever function was used. (Kind of bad practice as direct reference via function return value consumes more memory than a set variable. That being the case, I've used it sparingly and only when most efficient a.k.a. when it makes sense to do so rather than create a whole other variable just for a single-use purpose.)

I've included appropriate labeling for the Batch script as having at least the one label for the tokenization subroutine is required for usage from within the script itself. Granted, I (think) I could have used a nested for loop. However because I prefer being able to reuse the function in later loop-arounds anyway it's not much different. It would just take out the middle man of calling the label which, let's be honest, isn't going to tank performance of a Batch script, lol.

As you can see, and granted this is likely doable in many different forms and fashions across both languages, my renditions of the same script yielded a shorter Python script. Not by much, but shorter (and more or less easier to read/follow) nonetheless. That, and there's not a whole lot of confusing, not-so-easy-on-the-eyes inline variable expansion with additional call statements to add another layer of expansion on top of the existing. While I am well aware of the existence of Powershell, like I said I (and I would assume most others) would prefer to stick with what I know. Powershell is more versatile from what I remember, but not to the point at which Python is. I mean,

unless there is some huge underground Powershell community I'm not aware of. Additionally, compared to Python if I'm already needing some different syntax which I find more suitable and/or extended usability, I already know Python pretty well so why not use it? It's different from person to person, of course. I'm just one who prefers Python over Powershell (or at the very least is more leaning towards it if nothing else).

On a line-by-line comparison, Python did it on 20 lines while Batch did it in 27. Now, the tokenizing subroutine took a total of six lines. If Batch had the same function from Python, it'd have taken 22 lines. On top of that, there is no Python equivalent of "@echo off", so that'd be 21 lines. I actually surprised myself in doing this. If Batch had the same additional capabilities as Python, it'd have only taken (at best and being conservative) an additional one line to complete. I've known the capabilities of both languages for some time now, but damn.

Well, all that being said and done, I'm still open to opinions. That, and if you've any recommendations or changes to my scripts to shorten them (other than the aforementioned ridiculous shortening techniques) let me know. :)

HFDone - July 15, 2020, 7:44 am

python is great for writing quick scripts but you need to install it powershell came with windows
its very easy to convert it to powershell and im sure it will be shorter
i may do it if you really want

Edit - important thing about powershell is that you can easily use the power of NET framework that came with windows
you can take c# code and run it

```
Code $istream = "$pwd" + "\istream\  
$ostream = "$pwd" + "\ostream\  
$links = "_videolist.txt"  
$iext = ".webm"  
$oext = ".opus"  
$ytdl = "youtube-dl.exe --newline -f bestaudio -o "" + $istream + ""%(uploader)s - %(title)s.%(ext)s' -a  
"" + $links + ""  
cmd /c $ytdl  
Gci $istream -Recurse -Filter Topic - *.$iext | % {  
    $tmpfn = $_.Name -Replace "Topic - ", ""  
    $tokens = $tmpfn.Split(" - ")[0]  
    $command = "ffmpeg.exe -i "" + $istream + $_.Name + "" -vn -acodec copy -metadata artist="" +  
$tokens[0] + "" -metadata title="" + $tokens[1].Replace($iext, "") + "" "" + $ostream +  
$tmpfn.Replace($iext, $oext) + ""  
    cmd /c $command  
}
```

didnt debug and test it just to show you example

Sajuuk - July 15, 2020, 3:48 pm

(Jul 15, 2020 07:44 AM)HFDone Wrote: [\[CITE\]](#) python is great for writing quick scripts but you need to install it powershell came with windows

its very easy to convert it to powershell and im sure it will be shorter
i may do it if you really want

Edit - important thing about powershell is that you can easily use the power of NET framework that came with windows

you can take c# code and run it

That's actually been on my mind for some time now. I generally have a disliking for languages which don't run natively, but I like Python because of it's insane versatility and huge community.

That being the case, with Python I'm already using an interpreted language. Correct me if I'm misunderstanding the definitions, but because Powershell appears to run through the .NET framework (and as such requires it) I consider it interpreted just as Python is. Again, I might be confusing the definitions of managed code and interpreted code. But from what I can tell after reading up on Powershell and the WMF it requires to run, even though it doesn't say it installs the .NET framework explicitly I can't tell if some version of it is included by default and just not listed. (Or it may assume some appropriate version of the .NET framework is installed and throw an error in the case some outdated invalid version is found, who knows.)

Python is interpreted by its interpreter, while Powershell (to my knowledge) runs through the .NET framework. They both have middlemen, and at least somewhat similar syntax. Powershell does come native on newer OS versions, yes. But I guess I'm so accustomed to using Python that Powershell is out of the question at this point. As far as the C# capability within Powershell goes, I'd not be able to use it much unless I actually began delving into C#. I've never been the biggest fan of any of the .NET languages for reasons I won't go into as you can likely assume or have previously read what they are, lol. That, and... wait... if you can run C# code through Powershell doesn't that mean that the .NET framework is inherently required by default? I don't know why it took me so long to make that connection. Hell, if that is the case, I may as well continue using Python. I do appreciate Powershell's extended usability, but what more it offers in cases such as these (where the most needed is something like a Regex search/replace, which JREPL exists for when running "strictly" Batch) it doesn't outperform Python. And, to be honest, Python doesn't outperform Powershell nor Batch by very much in this scenario (especially now that I've completed the Batch script). It's namely just syntactic differences, so user preference goes above all.

Performance-wise, as I said in the OP, Python isn't going to be spectacularly faster from my point of view. Both Python and Batch run pretty well on-demand on both my old XP machine and my Windows 7 machine. That not being taken into consideration, I guess both Python and Powershell would perform in much the same fashion i.e. instant execution line by line. This is actually interesting to me quite a bit as for some time now I've been under the impression that Batch had some major faults when it came to versatility when compared to Python (additionally Powershell, even though I've never gone over it much). Writing up that little script yesterday has more or less changed my outlook on things, not to mention your example of the same script in Powershell pseudo which took a mere 13 lines (untested). I'll run through a few powershell topics and see about making the same in Powershell myself then run some comparisons after that.