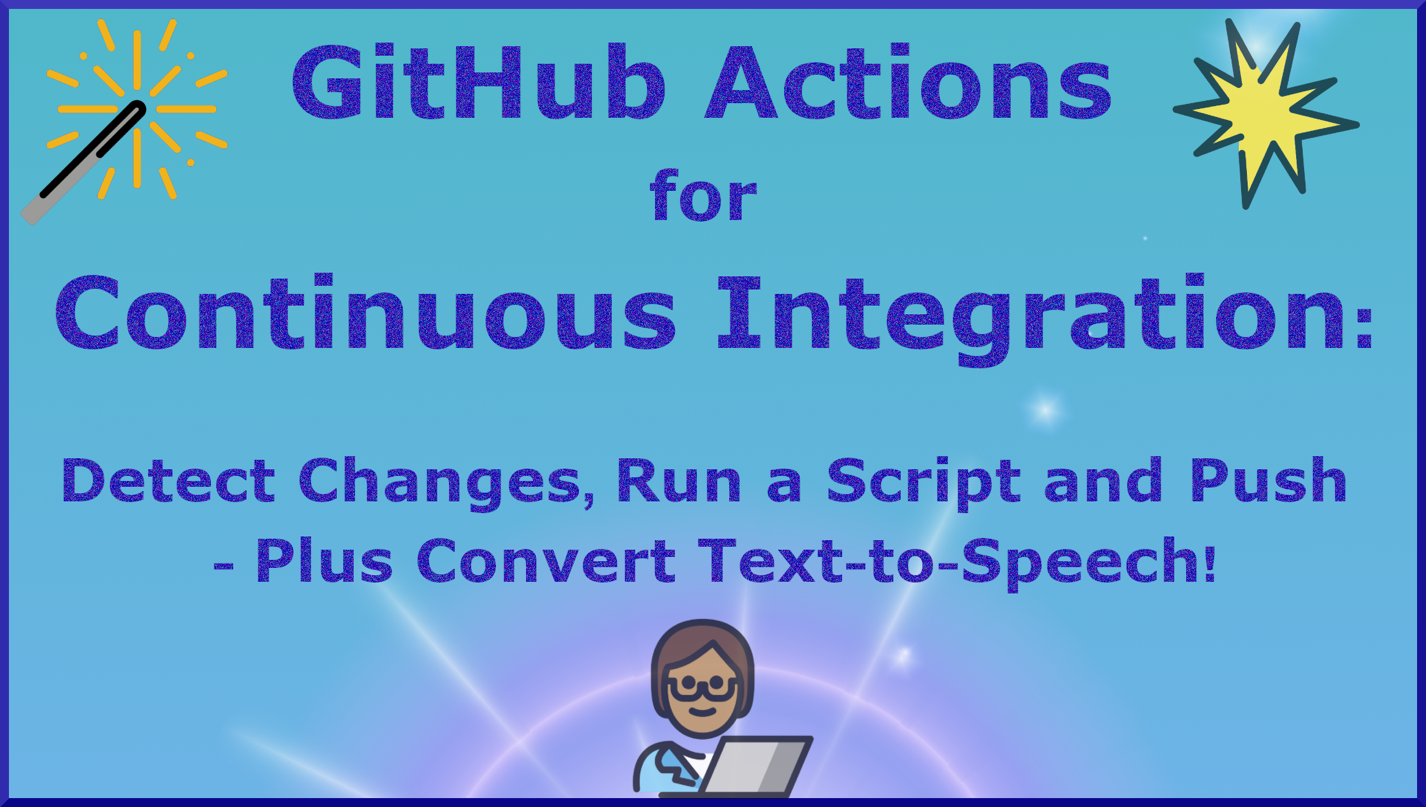
**GitHub Actions for Continuous Integration:**

**Detect Changes, Run a Script and Push – Plus Convert Text-to-Speech!**



The idea for this project came from our team’s desire to attach audio versions of the blogs and articles we post to the [Cisco Community](https://community.cisco.com/t5/technology-and-support/ct-p/technology-support). This will enable those who are visually impaired and give everyone else the option to listen instead of reading. I also wanted to have a place to store all my blog and article documents, other than a folder on my desktop. This source of truth (SOT) of text templates and their associated audio MP3s will also serve as an archive for future use of this knowledge.

The continuous integration (CI) pipeline which I’m about to share is worth sharing because it accomplishes two things in particular, which you may have assumed [GitHub Actions](https://docs.github.com/en/actions) would do out of the box, but for which I had to do some research to find and implement a niche tool.

These two things are:

1) Detect exactly which file(s) have changed in a Push to GitHub and assign the properties of those files to variables.

2) After running a script using those variables, automatically Commit and Push back into the GitHub repository.

Setting up the Directories in GitHub

We almost always post in the [Developer Hub](https://community.cisco.com/t5/developer-hub/ct-p/4409j-developer-home) area of the Cisco Community, so I just needed to model that structure in GitHub.

Here’s the contents of the Developer Hub, as seen online:

Graphical user interface, application

Description automatically generated

I created a new GitHub repository (repo) named “community-content-pipeline” and added a directory called “developer-hub” where I replicated the structure from the website.

Graphical user interface, text, application

Description automatically generated

Now, whenever I post a new article or blog in the Developer Hub, I will also upload my original docx document here, in the appropriate subfolder. Now, let’s automate some CI magic!

GitHub Actions Workflow

In order to setup a GitHub Action, we create a YAML file in a **/.github/workflows/** directory we create in our GitHub repo. Mine is laid out like this:

**community-content-pipeline/.github/workflows/make\_mp3\_on\_push.yml**

The first thing I did, besides naming it, was to set this workflow to only run on a PUSH to the repo and only if the files being pushed are located in the *developer-hub* directory and end in ‘.docx’ (this is the standard file format currently used by Microsoft Office – Word).

name: Create MP3 from .docx upon Push to a developer-hub directory

on:

push:

#only runs when files in this path are changed, and they must end in .docx

paths:

- developer-hub/\*\*.docx

Now, I needed to find a way to see exactly which file changed in the Push and assign it to a variable for working with it in the Python script we’ll compose later. This action does just the trick, and it’s found on GitHub Marketplace as well:

[**trilom/file-changes-action**](https://github.com/trilom/file-changes-action)

Here’s how I’ve used it in the workflow to see which files are added or modified and feed them to the Python script we’ll make.

- id: file\_changes # see which file changed in the push, in '' string output

uses: trilom/file-changes-action@v1.2.3

with:

outout: ''

fileOutput: ''

- name: test # print modified and added files

run: |

echo 'modified file(s) = ${{ steps.file\_changes.outputs.files\_modified}}'

echo 'added file(s) = ${{ steps.file\_changes.outputs.files\_added}}'

- name: execute py script # run file

env:

MODIFIED\_FILE: ${{ steps.file\_changes.outputs.files\_modified}}

ADDED\_FILE: ${{ steps.file\_changes.outputs.files\_added }}

run: |

python3 action\_txt\_2\_mp3.py

Ok, the workflow is almost done. Now, I needed to tell the workflow to commit the changes the Python script will make and push them back into the repo. Here’s how that is accomplished, thanks to the following GitHub Action, also found on the GitHub Marketplace:

[**ad-m/github-push-action**](https://github.com/ad-m/github-push-action)

- name: Commit files # transfer the new files back into the repository

run: |

git config --local user.name "xanderstevenson"

git add .

git commit -m "Updating the repository, with mp3 in the ./mp3s folder"

- name: Push changes # push the output folder to your repo

uses: ad-m/github-push-action@master

with:

github\_token: ${{ secrets.GITHUB\_TOKEN }}

force: true

Alright! We’re almost there. Now, we just need to write the Python script we’ve invoked in the workflow and we’ll be good to go!

Python Script to Convert Text-to-Speech

Remember the variables ‘MODIFIED\_FILE’ and ‘ADDED\_FILE’ we created in the workflow? Let’s import them into our Python script.

if \_\_name\_\_ == "\_\_main\_\_":

#import environmental variable from the GitHub actions workflow

# get the file that changed or was added and strip the brackets and quotes

the\_modified\_filename = os.getenv('MODIFIED\_FILE')

the\_modified\_filename = the\_modified\_filename.replace('[','').replace(']','').replace('"','')

the\_added\_filename = os.getenv('ADDED\_FILE')

the\_added\_filename = the\_added\_filename.replace('[','').replace(']','').replace('"','')

Now, we’ve got the name of the file that changed. But we still need its basepath as Python sees it. Here’s the method I used to do that:

base\_path = os.path.dirname()

We have the filename and basepath and now I want to create a directory named after the file that changed. This was how I got that to happen:

# get the filename of the document

file\_name = str(os.path.basename(txt\_filepath).rsplit(".", 1)[0])

file\_name = file\_name.replace(" ", "-")

txt\_dirname = mp3\_base\_path + f"/{file\_name}/"

if not os.path.exists(txt\_dirname):

os.makedirs(txt\_dirname)

Translating a docx to a MP3 was not working, so I found I had to convert the DOCX to a TXT first. I installed <docx2txt> for that and used it as follows. I also needed to format the resulting txt io remove URLs and empty lines, which could make the MP3 sound annoying when read in text-to-speech:

try:

# convert docx to txt

my\_text = docx2txt.process(docx\_filepath)

# remove all URLs from text

my\_text = re.sub(r"http\S+", "", my\_text)

# create and write text to txt file

with open(txt\_file, "w") as text\_file:

print(my\_text, file=text\_file)

#remove original docx

os.remove(docx\_filepath)

### removing empty lines

# Read lines as a list

workable\_txt = open(txt\_file, "r")

lines = workable\_txt.readlines()

workable\_txt.close()

# Weed out blank lines with filter

lines = filter(lambda x: not x.isspace(), lines)

# Write

workable\_txt = open(txt\_file, "w")

workable\_txt.write("".join(lines))

workable\_txt.close()

###

Finally, we feed the TXT file to the [gTTS](https://pypi.org/project/gTTS/) module I’ve installed and save the MP3 to a new sub-directory we create called ‘mp3s.

# open and read .txt file

with open(txt\_file, 'r') as f:

the\_text = f.read()

# conversion to MP3

mp3 = gTTS(the\_text, lang="en", tld=accent)

# strip filename from filepath

file\_name = str(os.path.basename(txt\_file).rsplit('.', 1)[0])

# if the 'mp3s' directory does not exist, create it

if not os.path.exists(txt\_dirname + '/mp3s/'):

os.makedirs(txt\_dirname + '/mp3s/')

time.sleep(3)

# name and save mp3

mp3\_filename = txt\_dirname + '/mp3s/' + file\_name

mp3\_filename += '.mp3'

mp3.save(mp3\_filename)

At this point in the workflow, the following will be committed and pushed to the repo:

* A new folder, named for your .docx file, inside of which will be:
  + A downloadable copy of the original DOCX document
  + A readable TXT version of the document
  + A downloadable MP3, inside of an 'mp3s' sub-folder

Demo

Let’s see this baby in action! I have a DOCX document, named “DevSecOps-Defined”, which contains text, images, links and URLs.

[Graphical user interface, text, application

Description automatically generated](https://community.cisco.com/t5/security-knowledge-base/devsecops-defined/ta-p/4712801)

To read the DevSecOps Defined article, just click on the image.

All I need to do is upload and commit it to the appropriate directory in my repo, which in this case is **community-content-pipeline/developer-hub/developer-security/security-knowledge-base/**

That will start the build process that is detailed in our Workflow. GitHub Actions does the rest.

[insert Pipeline-Build video]

For GitHub Actions, the entire Build, Test and Report process for CI took about 30 seconds; not too bad! Here are the results: a new directory named “DevSecOps-Defined” with our DOCX and TXT files inside….



….as well as the *mp3s* folder with the audio inside.

Graphical user interface, text, application

Description automatically generated

Source Code, Related Projects and Thanks

For the complete code to this project, see the following repository:

Shape

Description automatically generated

<https://github.com/xanderstevenson/community-content-pipeline>

People may want control over the accent used for text-to-speech. For that, I have created the following:

Icon

Description automatically generated

<https://github.com/xanderstevenson/txt_2_mp3>

This will give you the ability to choose from the following accents for the MP3:

1. English (Australia)

2. English (United Kingdom)

3. English (United States)

4. English (Canada)

5. English (India)

6. English (Ireland)

7. English (South Africa)