It's been a long time since I made a commit with my main Github account at @mazipan, that's because I decided to create a new account to separate my work from Github which I intended to explore things that are usually outside of work. This keeps my stats on Github from turning green anymore ��.
Since I got to know Github Actions , I immediately fell in love with this feature from Github. I usually take advantage of various continuous integration (CI) tools such as Tracis, Circle CI and so on to automate my work, when I was hooked with this feature. I can manage the main code and CI system in one place, no longer need to move between websites to manage everything. Another thing also makes me believe that Github Actions is quite promising, because it was developed by the Github team themselves so they are the ones who I think understand best how best to integrate various needs with their platform.
Sure enough, once, twice and several times I tried to read and apply Github Actions to my repository, how come it's really good. On a whim I thought what if I just made a scheduling code that would do simple commits automatically to my own Github account. So cring... I started blusukan for imaging and looking for reading material on how I can do a reverse commit to the repository that I'm currently using in Github Actions.
Coincidentally, this topic is right where I need to create automation for the home page repository belonging to Vue.js Indonesia, they use two repositories where one repository is the source code and the other repository is a repository for the results that have been processed by VuePress.
Neither easy nor difficult. Difficult, because I've never worked on this model. It's easy, after doing it (oh, it's just like this anyway ��).

Memulai Github Actions

Github Actions configuration files use YAML files and are located in a `.github/workflows` common directory. In this simple example, I created a `.github/workflows/autocommit.yml`. We can add more than one file in this directory usually if we have some work that are not related to each other, to make it easier to organize them in the future.
First we need to add the name of our workflow, so I just need to add the code inside the YAML file like this:

```
name:Auto commit
```

Get to know Triggers

In my opinion, the basic things that must be known to make CI are understanding triggers and jobs. Trigger is the code that determines when our CI will be executed, while Job is the job that will be executed when the rules in the trigger have been fulfilled. In Github Actions, this Trigger is known by the name Eventsin Github Actions and is marked with the syntax onor simple language "what does this fit?". so what else do you want this code to run. Because I want to do a test first, I add Trigger just right again push. I can add the following code:

```
name:Auto commit
on:
push:
branches:
-master
```

You can listen to the code again if it's still floating around, the code above can be explained that I added a trigger when something happened on my pushbranch . masterSo every time push, Auto committhis workflow will run. But actually this is not what I want to achieve, I want to make a scheduling code so that I don't have to be manual pushall the time. Therefore I added another Trigger which will run automatically when the specified time is reached. I added the following code:

```
name:Auto commit
on:
push:
branches:
-master
schedule:
-cron:'0 7 * * *'
```

Can we review the code again? In the above code I can explain that I want this workflow to run every day at 7 am. You see I added below onparallel to pushie schedulewhich means to schedule a job. What schedule to use, the easiest and most popular is to use the cron string. This is a kind of text that

we can use to determine a schedule. I usually use the crontab.guru web if I still have doubts about the text that I have created.

Adding tasks to Github Actions

Adding a task or job that will be executed when the trigger is met is the main part which is usually easy and difficult. First we add a keyword `jobs`that indicates the start of a block for the task to be executed. Underneath we we can give a name to our job, I give the name `auto_commit`. In one workflow and one trigger, we can add more than one job. the next step we can define sequential steps that must be done from the job we want to create. Here's a simple code example to create a job on Github Actions:

```
name:Auto commit
on:
push:
branches:
-master
schedule:
-cron:'0 7 * * *'
jobs:
auto_commit:
runs-on:ubuntu-latest
steps:
-name:Test the jobs
run:|
          echo "just another test"
```

I added a step for testing only, just defining that this task will run on the latest ubuntu operating system and providing a step where it just prints the word “just another test”.
At this point, you can already commit and push to `master`your branch to check whether the workflow we created has executed correctly according to our wishes or not. To check, after our code has been pushed, we can visit any tab `Actions`in our repository (example: https://github.com/mazipan/auto-commit/actions) and we can see a list of workflows that have been or are being executed.

Checkout git repository

The first step I have to do when I have the goal of making an automatic commit is to be able to checkout the code from the git repository that we are currently working on. One of the cool things about Github Actions is the model that can be plugged in from various Actions that have been published, both official ones from Github and from individuals. Just like in JavaScript, which makes use of the many libraries that have been published to make our task easier. To do self- checkout , there are already official Actions from Github and we just need to use them. Here's an example of the code:

```
name:Auto commit
on:
push:
branches:
-master
schedule:
-cron:'0 7,8,9,10,11 * * *'
jobs:
auto_commit:
runs-on:ubuntu-latest
steps:
-uses:actions/checkout@v2
with:
persist-credentials:false
fetch-depth:0
```

Just add usesand use the name of the published Actions. The actions we use are from https://github.com/actions/checkout . Underneath is with, these are the parameters that we want to pass together when calling the checkout Actions. Two parameters persist-credentialsdan fetch-depthare needed to ensure a successful reverse commit due to an issue related to this in Actions which we will use later. We will explain in the next section.

Make sure there are changes every time

In order to be able to commit and push back to the repository that we are currently using, we must ensure that there are changes (changes) that occur in one or several of our files. To get around this, I created a file that I will update every time this workflow is run. I created a file LAST_UPDATEDthat doesn't need to contain anything at this point. Back in our workflow, I added simple code to manipulate the contents of the file we just created into one of the steps in our workflow. Here's the code:

```
name:Auto commit
on:
push:
branches:
-master
schedule:
-cron:'0 7,8,9,10,11 * * *'
jobs:
auto_commit:
runs-on:ubuntu-latest
steps:
-uses:actions/checkout@v2
with:
persist-credentials:false
fetch-depth:0
-name:Modify LAST_UPDATED file
run:|
          d=`date '+%Y-%m-%dT%H:%M:%SZ'`
 echo $d > LAST_UPDATED
```

You can see that I only filled in the file LAST_UPDATEDwith the date when running the workflow.

Commit and push back

If it's local, we usually after making changes we can use the `git commit -m "sebuah pesan"`later command `git push origin master`to finish the process. The workflow is basically the same, it's just that we will use the Actions from ad-m/github-push-action to push back to our repository. First, we'll commit with the following code:

```
name:Auto commit
on:
push:
branches:
-master
schedule:
-cron:'0 7,8,9,10,11 * * *'
jobs:
auto_commit:
runs-on:ubuntu-latest
steps:
-uses:actions/checkout@v2
with:
persist-credentials:false
fetch-depth:0
-name:Modify LAST_UPDATED file
run:|
          d=`date '+%Y-%m-%dT%H:%M:%SZ'`
 echo $d > LAST_UPDATED
-name:Commit changes
run:|
          git config --local user.email "{YOUR_EMAIL}"
          git config --local user.name "{YOUR_USERNAME}"
 git add -A
 git commit -m "Sebuah pesan"
```

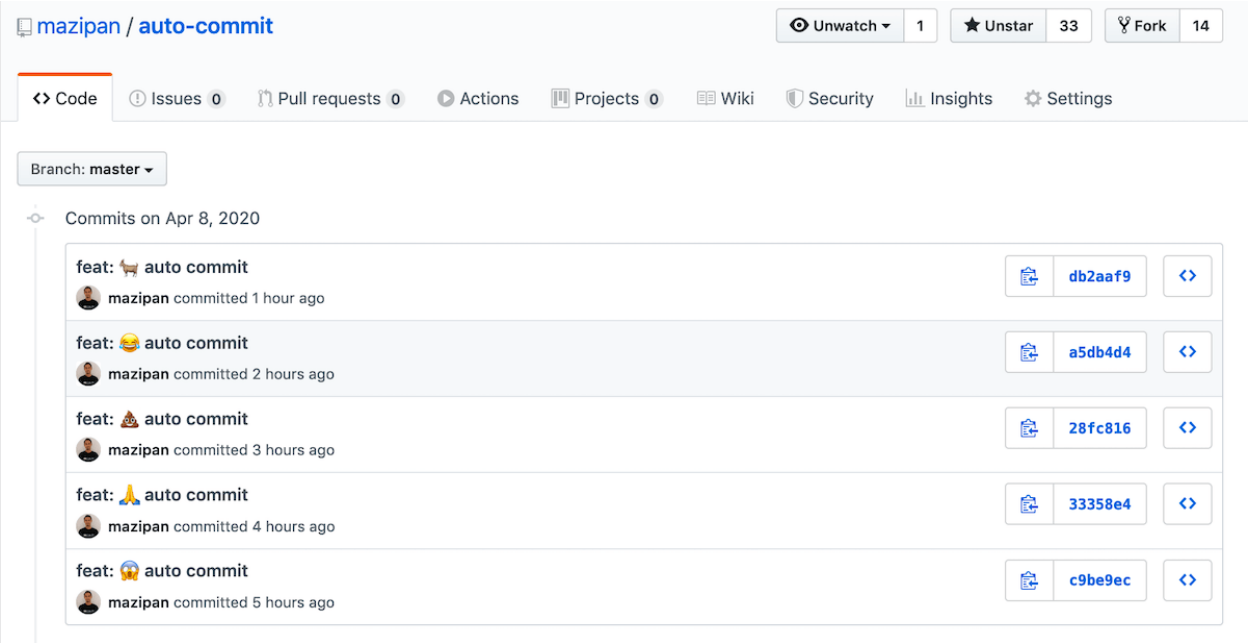
The above code commits with the previous we set with the email and name we want. Next add a new step to commit back to our repository, here is the sample code:

```
name:Auto commit
on:
push:
branches:
-master
schedule:
-cron:'0 7,8,9,10,11 * * *'
```

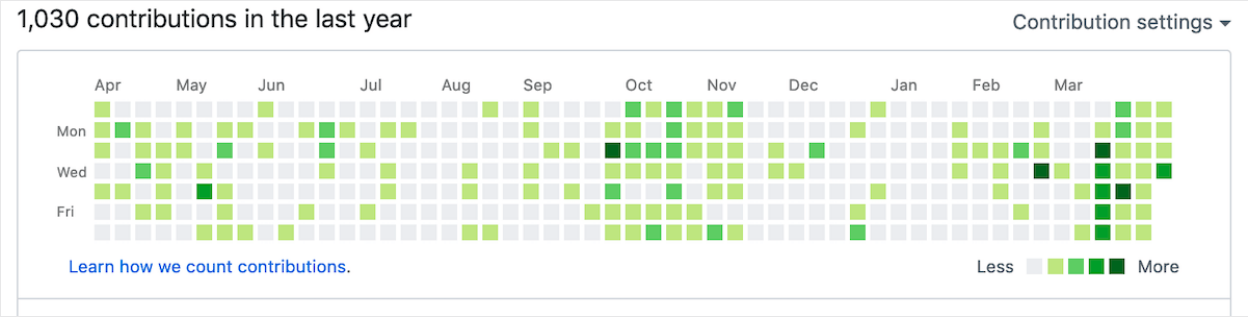
```
jobs:
auto_commit:
runs-on:ubuntu-latest
steps:
-uses:actions/checkout@v2
with:
persist-credentials:false
fetch-depth:0
-name:Modify LAST_UPDATED file
run:|
          d=`date '+%Y-%m-%dT%H:%M:%SZ'`
 echo $d > LAST_UPDATED
-name:Commit changes
run:|
          git config --local user.email "{YOUR_EMAIL}"
          git config --local user.name "{YOUR_USERNAME}"
 git add -A
 git commit -m "Sebuah pesan"
-name:Push Back
uses:ad-m/github-push-action@v0.5.0
with:
force:true
directory:'.'
github_token:${{ secrets.GITHUB_TOKEN }}
```

We use Actions ad-m/github-push-action@v0.5.0by adding a parameter forceto force push , directorywith a point value which means that all changes from all directories will be pushed back, and github_tokenby utilizing the standard token that has been implanted by Github Actions so we don't need to add our personal access token manually.
Related to the checkout step in the previous section which requires adding additional parameters related to issue #44 which is in the repo ad-m/github-push-actionwhich has been answered by the author himself.

How about the results?



Auto commit message



Github stats

Yes, you can basically change the configuration yourself how many times a day this auto-commit runs. In my case, 5 times is enough. It's not very green, but it's pretty good it's not dry ��

Sumber repository

All of this writing, is based on the repository which you can see at the following link:
https://github.com/mazipan/auto-commit/

Thank you and hopefully not abused ��