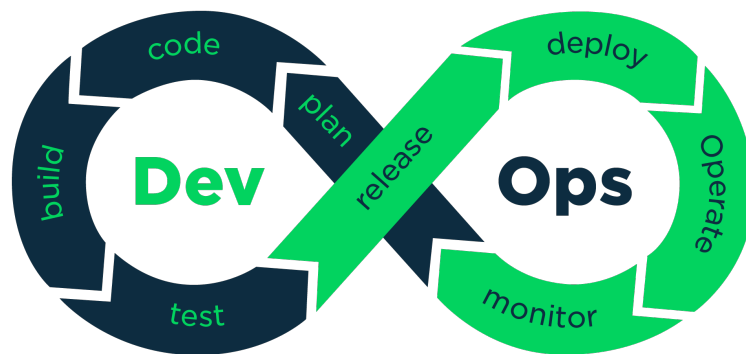


B2 - Introduction to DevOps

B-DOP-200

Chocolatine

Improve integration and testing with GitHub Actions



Chocolatine



- The totality of your source files, except all useless files (binary, temp files, obj files,...), must be included in your delivery.



Whatever your opinions might be, this is a *chocolatine*.

But what purpose does it serve? It allows you to go through the day and through your work better. In other words, it facilitates your work and allows you to focus on things that matter (discussing its name does not pertain to this category).

Your repositories are hosted on GitHub, and GitHub has an intuitive way of automating the launch of pre-defined actions, triggered by events of your choice. It is, in the same way as chocolateines, a great way to facilitate and enhance your development and your workflow.

In this project, you will setup a GitHub Actions workflow to enforce good practices and diverse rules in a repository.



GitHub Actions are free and unlimited for public repositories, and free up to 2,000 minutes per month for personal private repositories, which is quite generous!



TECHNICAL DETAILS

You will have to turn in a single YAML workflow file named `chocolatine.yml`, placed at the root of your repository.

Your workflow must be usable with your different Epitech projects and their respective technologies and associated tools.



For this project, the only external actions allowed are `actions/checkout` and `pixta-dev/repository-mirroring-action`. All other external actions (which you can find on the GitHub marketplace for example) are strictly forbidden.



If particular settings or elements are not specified or addressed in the subject, you are free to do as you please with them.

EVALUATION

Your workflow will be tested by copying your `chocolatine.yml` file, **and only this file**, into a test repository's `.github/workflows` directory.

Make then sure that the workflow is self-contained and does not need any external files.

SECRETS

You might need to use secured data or values to make your workflow successfully run. In this case, you **must** use secrets.



If any hardcoded sensitive value is found in your workflow file, your entire project will fail. You have been warned.



FEATURES

Using a GitHub Actions workflow, you will need to implement a set of features.

The workflow must be run on every push and on every pull request creation, **unless** the branch name starts with `ga-ignore-`, in which case the workflow must not be run at all.

Furthermore, **each job must**:

- start by **checking out the repository** to the relevant branch;
- only be run if the **previous job has succeeded**.



The order in which the jobs are expected to be run is the order in which they are defined below in the subject.

No matrixes are to be used in this project.

ENVIRONMENT VARIABLES

You need to define several environment variables, available at the workflow level.

- `MIRROR_URL`: the URL of the Epitech repository which will act as mirror;
- `EXECUTABLES`: a comma-separated list of the paths of the executables expected to be produced (it can have a single executable, like `"pushswap"` or `"lib/my/libmy.a"`, or several, like `"antman/antman,giantman/giantman"`).



All environment variables' values will be strings.



“available at the workflow level” means that the environment variables must be only defined once, and be available for all jobs and steps of the workflow.

JOBS

CHECKING THE REPOSITORY CLEANLINESS

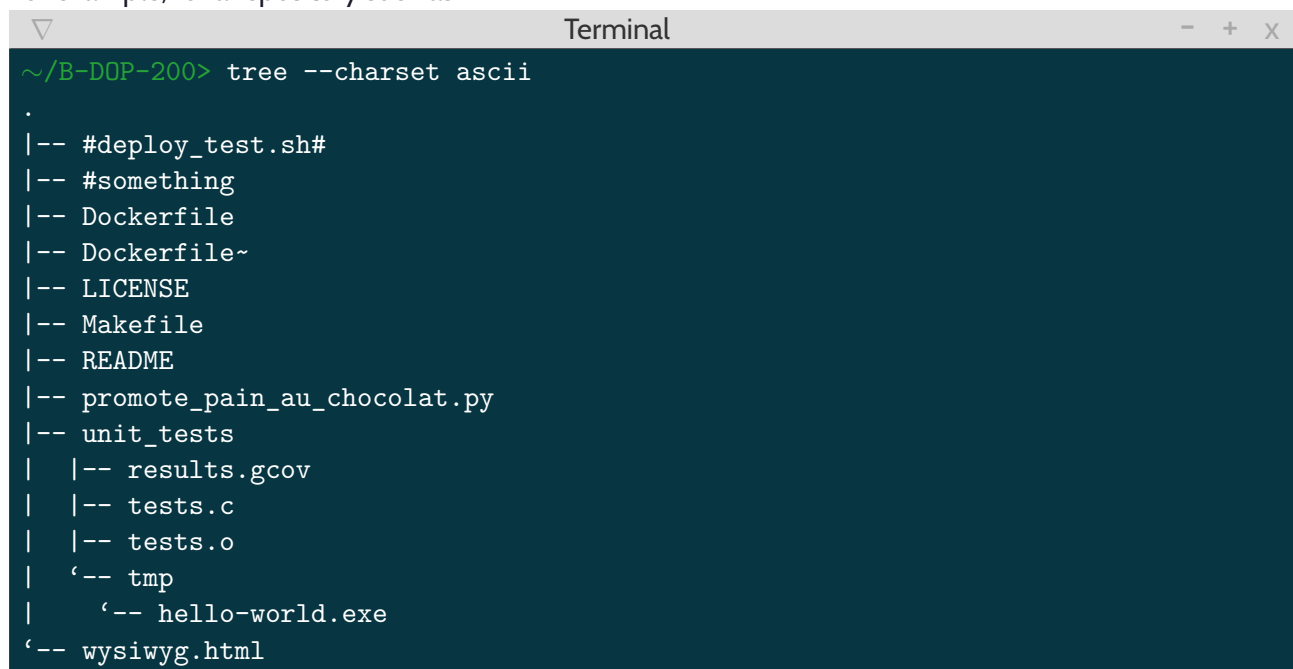
Create a job with a `check_repository_cleanliness` ID that, in order:

- finds a list of all unwanted files in the repository;
- if any, displays each unwanted file as an error annotation (which, if clicked on, must take the user to said file);
- makes the job fail if there are any unwanted files.

An unwanted file is any file or directory that matches at least one of these criteria:

- name starts and ends with a #;
- name ends with a ~;
- name contains `pain_au_chocolat` (case insensitive);
- has a `.o`, `.a`, `.so`, `.gcno`, `.gcda` or `.gcov` extension;
- is in a directory (or subdirectory) named `tmp`.

For example, for a repository such as:



```
~/B-DOP-200> tree --charset ascii
.
|-- #deploy_test.sh#
|-- #something
|-- Dockerfile
|-- Dockerfile~
|-- LICENSE
|-- Makefile
|-- README
|-- promote_pain_au_chocolat.py
|-- unit_tests
|   |-- results.gcov
|   |-- tests.c
|   |-- tests.o
|   '-- tmp
|       '-- hello-world.exe
'-- wysiwyg.html
```



The output of the job must be:

```
Annotations
7 errors

✗ Unwanted file detected: Dockerfile~#L1
Dockerfile~

✗ Unwanted file detected: unit_tests/tmp/hello-world.exe#L1
unit_tests/tmp/hello-world.exe

✗ Unwanted file detected: unit_tests/results.gcov#L1
unit_tests/results.gcov

✗ Unwanted file detected: unit_tests/tests.o#L1
unit_tests/tests.o

✗ Unwanted file detected: promote_pain_au_chocolat.py#L1
promote_pain_au_chocolat.py

✗ Unwanted file detected: #deploy_test.sh##L1
#deploy_test.sh#

✗ Checks if the repository is clean and void of any unwanted files (temp
files, binary files, etc.)
Process completed with exit code 1.
```



The order in which you display the unwanted files does not matter.



The last element in the above screenshot is the annotation automatically made by GitHub to signify that the job has failed. You can use the display name you want for your jobs, the one used on the above example is just an example.



CHECKING THAT THE PROGRAM COMPILES CORRECTLY

Create a job with a `check_program_compilation` ID that, in order:

- launches `make` at the root of the repository (this step must have a 2 minutes timeout);
- launches (on a separate step) `make clean` at the root of the repository;
- verifies that each file specified in the `EXECUTABLES` environment variable exists and is executable (the job must fail if any of the executables is not there, or not executable).

This job must must be run in the `epitechcontent/epitest-docker` Docker image.

RUNNING TESTS (BECAUSE THAT'S WHAT HEROES DO)

Create a job with a `run_tests` ID that launches `make tests_run` at the root of the repository (this step must have a 2 minutes timeout).

This job must must be run in the `epitechcontent/epitest-docker` Docker image.

PUSHING TO THE MIRROR REPOSITORY

Create a job with a `push_to_mirror` ID that runs a mirroring to the repository specified in the `MIRROR_URL` environment variable.

It must use a secret named `GIT_SSH_PRIVATE_KEY` to specify the SSH private key to use.

This job must only be executed when a push is made to the repository.