EPITECH
WORKSHOP

GitHub Actions

Vincent ANDRIEU

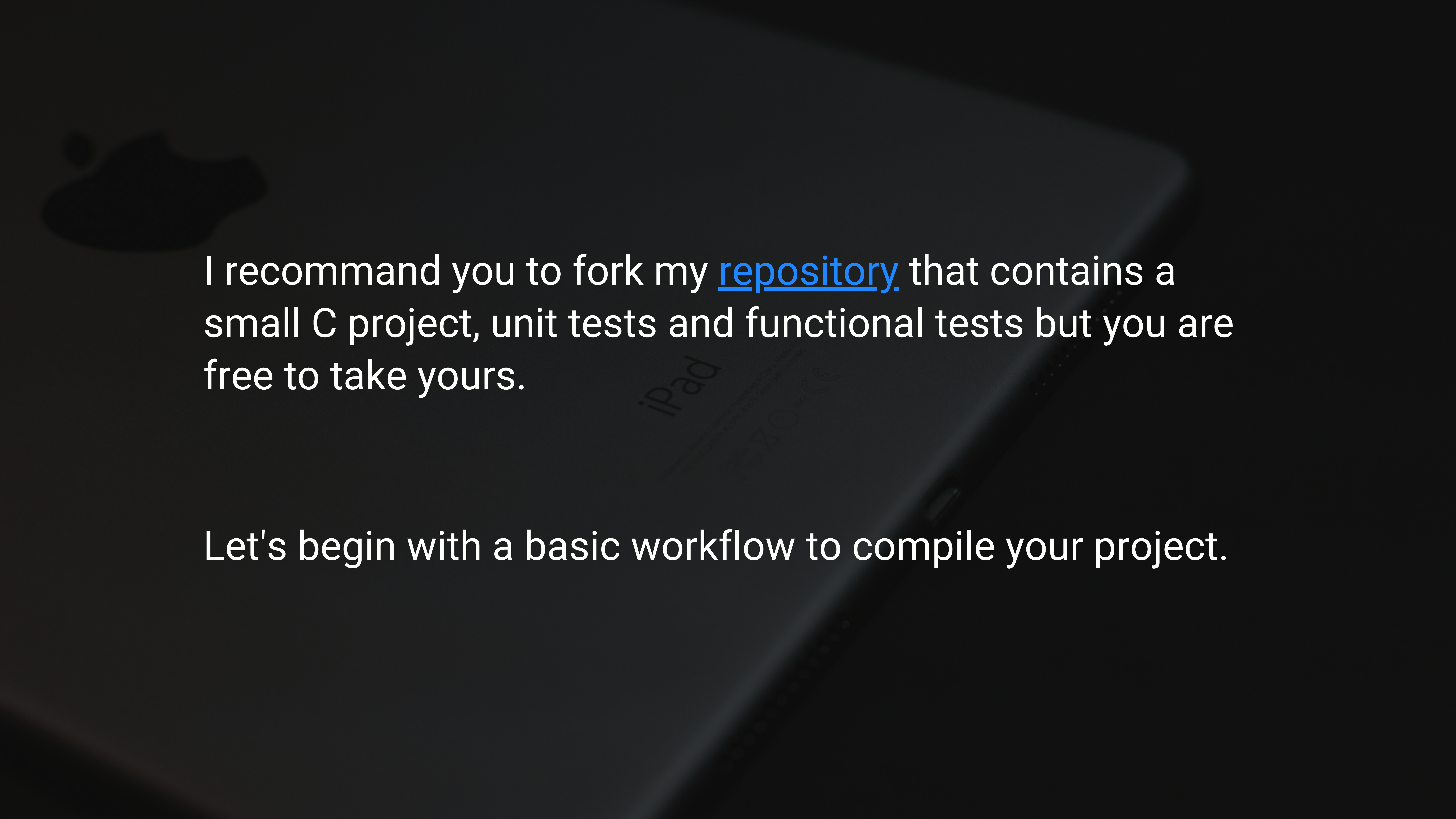
What are GitHub Actions ?

- CI/CD
- Implemented the November 13 2019
- On any OS
- Runnable with docker image
- For any language

Today's tasks

3 Workflows

1. Check Compilation
2. Run Unit tests
3. Run Functional tests
4. Check coding style
5. Go further



I recommand you to fork my repository that contains a small C project, unit tests and functional tests but you are free to take yours.

Let's begin with a basic workflow to compile your project.

Compilation

Firstly, create a workflow as a YAML file in a *.github/workflows* directory at the root of your repository. A workflow is automatically triggered.

This workflow will allow you to test the project compilation.
Syntax documentation

1	**Name**	Check compilation
2	**Trigger event**	Push on master
3	**Job name**	compilation
4	**Runs on**	Ubuntu 20.04
5	**Docker image**	You can use the same environment as the *moulinette :* epitechcontent/epitest-docker:devel

Compilation steps

Each workflow step is defined by a name and a command or an action.

You can find actions done by the community to simplify your workflow.
GitHub Actions Marketplace
Or you can write it by yourself.

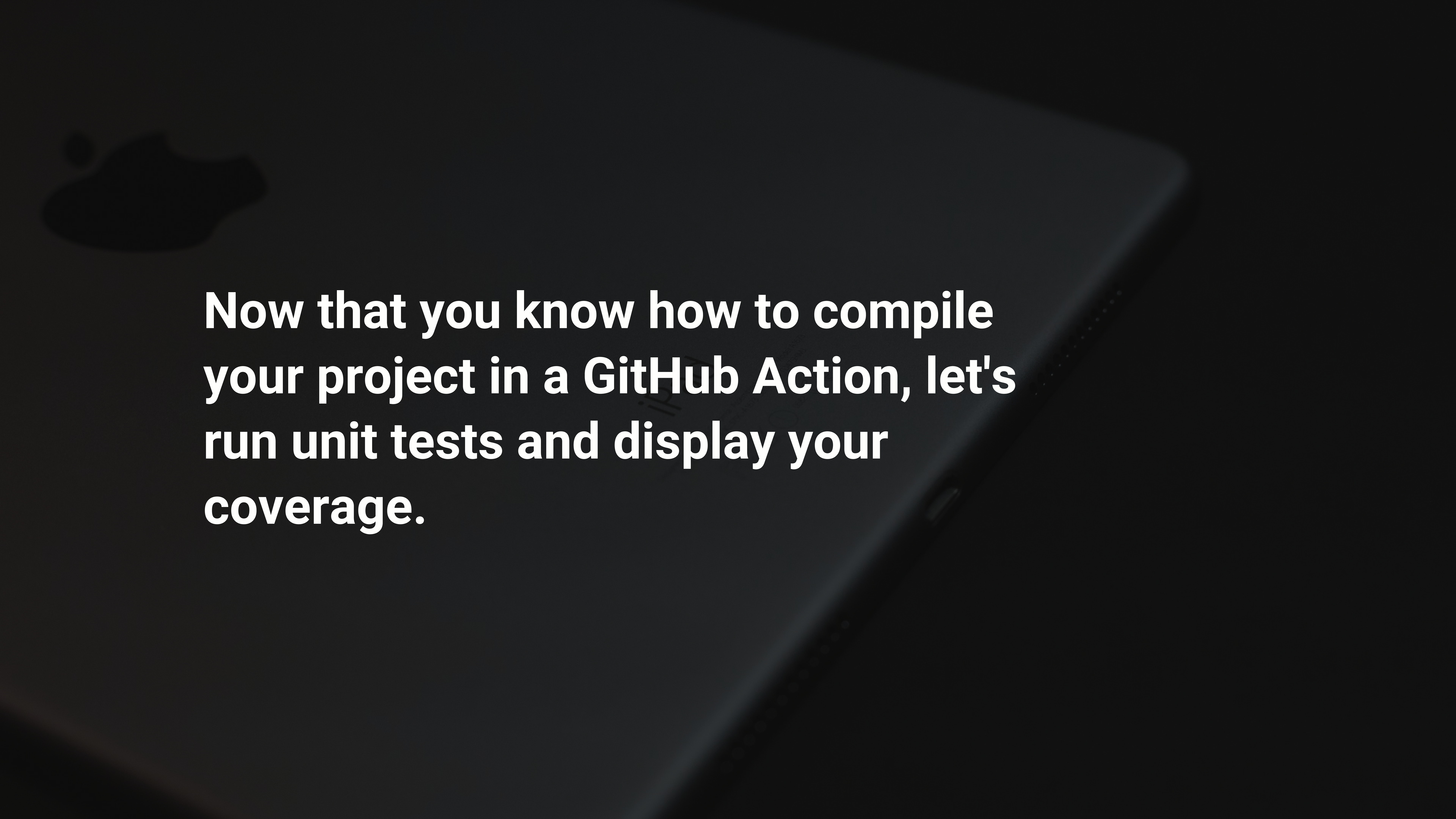
6 **Checkout your repository**

7 **Build your project** Write commands in different steps as you can do on your computer.

8 **Check if the result binary file exist**

9 **Don't run the binary** It's just a compilation workflow.

10 **Test** Push on master and check on GitHub in the *Actions* tab if the workflow success.



Now that you know how to compile your project in a GitHub Action, let's run unit tests and display your coverage.

Unit tests

Create a new workflow to run unit tests and display your project coverage.

1	**Name**	Run tests
2	**Trigger event**	When the *Check compilation* workflow is completed successfully
3	**Job name**	unit-tests
4	**Runs on**	Ubuntu 20.04
5	**Docker image**	epitechcontent/epitest-docker:devel

Unit tests steps

A workflow fails if a command exits with anything other than 0.

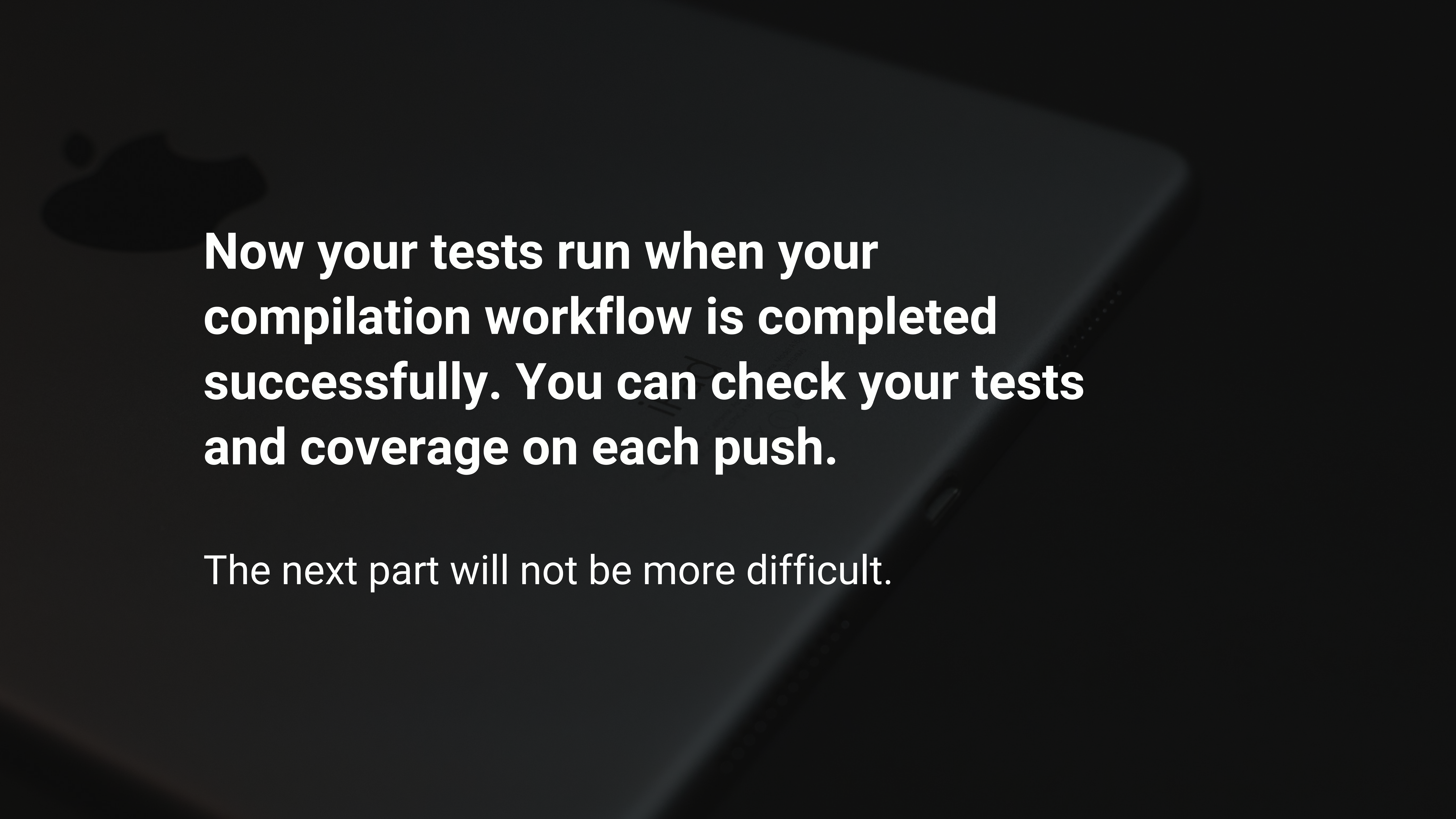
6 **Checkout your repository**

7 **Run the tests**

8 **Display your coverage**

9 **Test**

If tests fail, workflow must fail too.



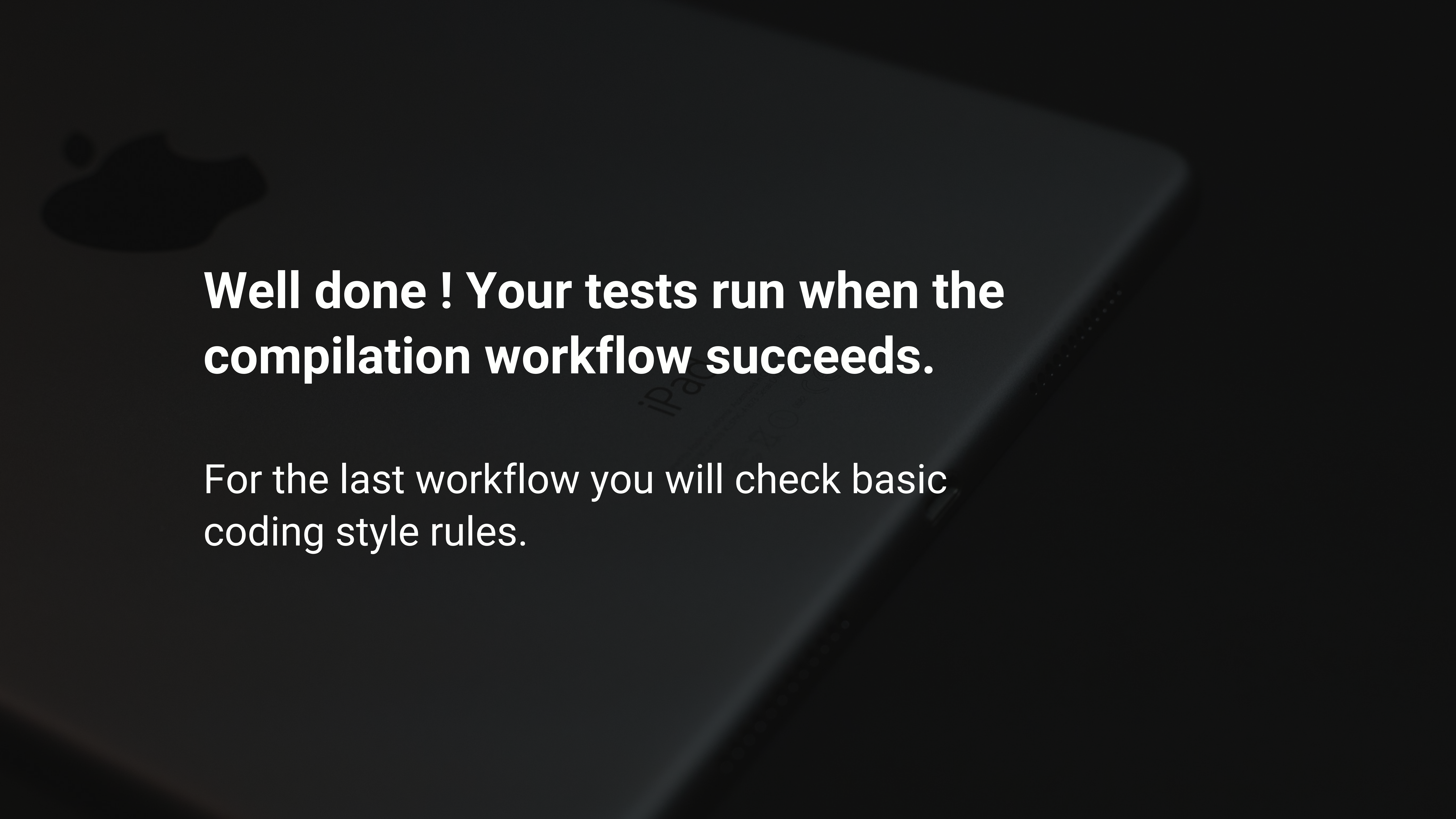
Now your tests run when your compilation workflow is completed successfully. You can check your tests and coverage on each push.

The next part will not be more difficult.

Functional tests

Add a new job to the previous workflow.

1	**Job name**	functional-tests
2	**Build the project**	
3	**Run the tests**	
4	**Test**	If tests fail, workflow must fail too.



Well done ! Your tests run when the compilation workflow succeeds.

For the last workflow you will check basic coding style rules.

Coding style

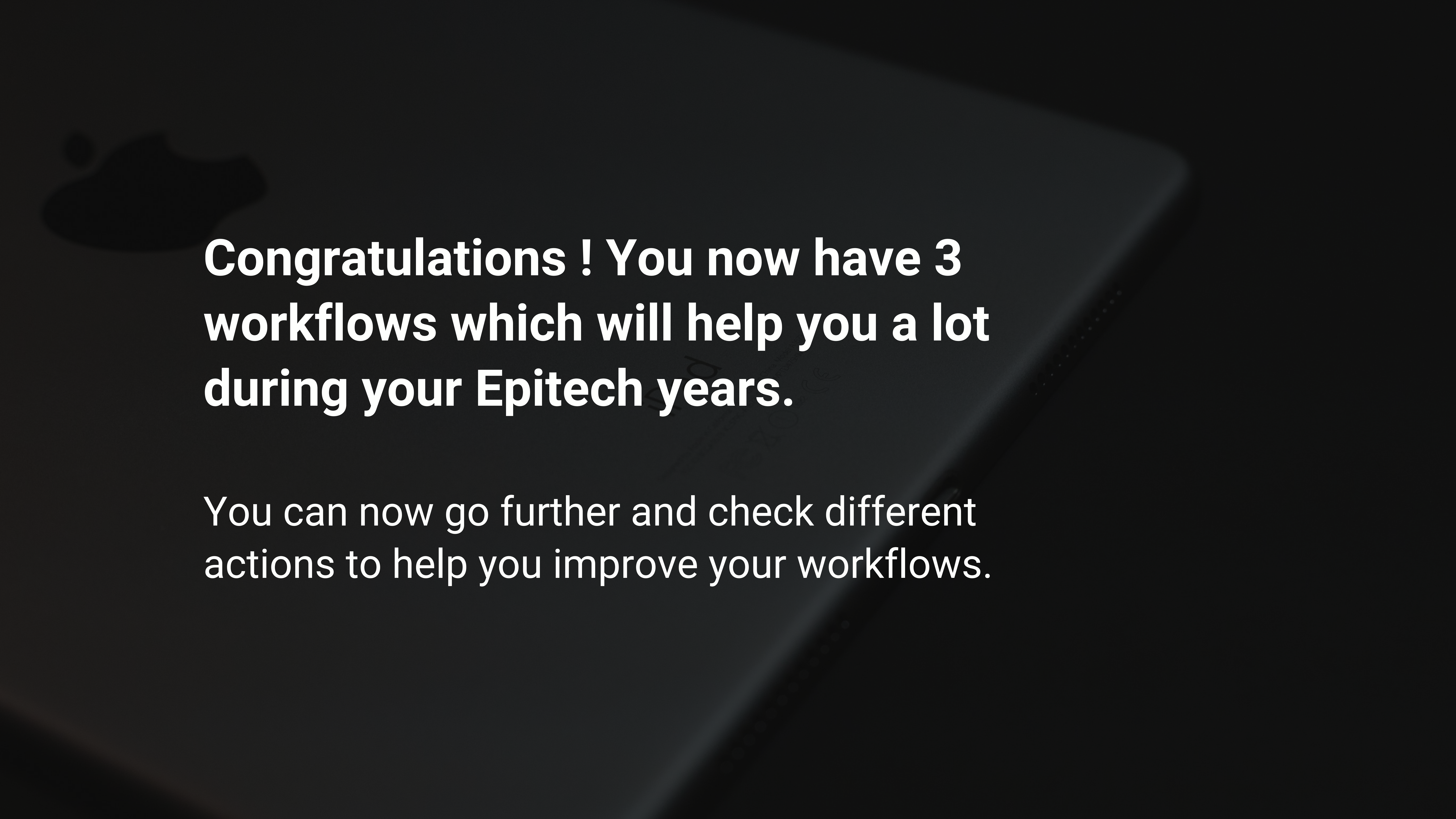
Create a new workflow to check basic coding style rules.

1	**Name**	Check coding style
2	**Trigger event**	On push on master
3	**Runs on**	Ubuntu 20.04 We don't need to run on *moulinette* environment.

Coding style

- **Tip:** Check *actions/checkout* documentation.
- **Reminder:** Check GitHub Actions syntax documentation
- **Reminder:** A step fails if the command returns anything other than 0.

4	**Checkout your repository**	In a directory
5	**Checkout NormEZ repository**	In a different directory ronanboiteau/NormEZ
6	**Check errors**	Create a step to each NormEZ errors and grep on them to check if it finds them.
7	**Continue even if it find an error**	This workflow should continue even if a coding style error is found but still mark the workflow as fail if it found one.
8	**Test**	Test if the workflow fails if there is a coding style error (it should flag multiple steps as failed if there are multiple different errors) in the project and success otherwise.



Congratulations ! You now have 3 workflows which will help you a lot during your Epitech years.

You can now go further and check different actions to help you improve your workflows.

Advanced features

You can choose one of these example features and try to implement them in a workflow.

JOB DEPENDENCIES

Why ?: To run a job after another one.

Example: You have multiple projects and one depends on another.

Link: https://docs.github.com/en/actions/learn-github-actions/workflow-syntax-for-github-actions#jobsjob_idneeds

CACHE

Why ?: To improve your workflow speed.

Example: Caching C++ build folder or conan data.

Link: https://github.com/actions/cache

ARTIFACTS

Why?: To upload files or directories to the current run.

Example: Upload unit tests coverage result.

Link: https://docs.github.com/en/actions/advanced-guides/storing-workflow-data-as-artifacts

Thank you for attending this workshop !

Good luck with GitHub Actions 😉