**CONTRIBUTING.md:**

The CONTRIBUTING.md is used to describe the process for contributing to the repository. A link to the CONTRIBUTING.md file is shown anytime someone creates a new issue or pull request.

**ISSUE\_TEMPLATE.md:**

The ISSUE\_TEMPLATE.md is another file you can use to pre-populate the body of an issue. For example, if you always need the same types of information for bug reports, include it in the issue template, and every new issue will be opened with your recommended starter text.

**ISSUE:**

a place where you can have conversations about bugs in your code, code review, and just about anything else

**Repositiory notifications: Watch**

You'll receive a notification when a new issue, pull request or comment is posted, and when an issue is closed or a pull request is merged. **Not watching**: You'll no longer receive notifications unless you're @-mentioned. **Ignore**: You'll no longer receive any notifications from the repository

**create a new repository on the command line**

echo "# Book\_Samples" >> README.md

git init

git add README.md

git commit -m "first commit"

git branch -M main

git remote add origin <https://github.com/dimescc/Book_Samples.git>

git push -u origin main

### push an existing repository from the command line

git remote add origin <https://github.com/dimescc/Book_Samples.git>

git branch -M main

git push -u origin main

**Clone repository:**

In der Commandozeile mit:

git clone <https://github.com/dimescc/python-random-quote.git>

Oder per SSH in Git direkt.

### Tips for using branches

A single project can have hundreds of branches, each suggesting a new change to the main branch.

The best way to keep branches organized with a team is to keep them concise and short-lived. In other words, a single branch should represent a single new feature or bug fix. Creating a branch allows you to make modifications to your project without changing the deployed main branch. Now that you have a branch, it’s time to create a file and make your first commit!

### Pull request

Now that you’ve created a commit, it’s time to share your proposed change through a pull request! You have created a branch, added a file, and committed the file to your branch. Now it’s time to collaborate on your file with other students taking this class. This collaboration happens in a pull request. This pull request is going to keep the changes you just made on your branch and propose applying them to the main branch.

* From the "Pull requests" tab, click **New pull request**
* In the "base:" drop-down menu, make sure the "main" branch is selected
* In the "compare:" drop-down menu, select "my-slide"
* When you’ve selected your branch, enter a title for your pull request. For example Add dimescc's file
* The next field helps you provide a description of the changes you made. Feel free to add a description of what you’ve accomplished so far. As a reminder, you have: created a branch, created a file and made a commit, and opened a pull request
* Click **Create pull request**

You successfully created a pull request, and it has passed all of the tests. You can now merge your pull request.

1. Click **Merge pull request**
2. Click **Confirm merge**
3. Once your branch has been merged, you don't need it anymore. Click **Delete branch**.

### GitHub Actions:

GitHub Actions are a flexible way to automate nearly every aspect of your team's software workflow. Here are just a few of the ways teams are using GitHub Actions:

* Automated testing (CI)
* Continuous delivery and deployment
* Responding to workflow triggers using issues, @ mentions, labels, and more
* Triggering code reviews
* Managing branches
* Triaging issues and pull requests

Actions come in two types: **container actions** and **JavaScript actions**. Actions come in two types: **container actions** and **JavaScript actions**.

**Create a dockerfile in folder /action-a/Dockerfile:**

FROM debian:9.5-slim

ADD entrypoint.sh /entrypoint.sh

RUN chmod +x /entrypoint.sh

ENTRYPOINT ["/entrypoint.sh"]

**Crate Script entrypoint.sh:**

#!/bin/sh -l

sh -c "echo Hello world my name is $INPUT\_MY\_NAME"

**Next, we'll define the action.yml file which contains the metadata for our action.**

name: "Hello Actions"

description: "Greet someone"

author: "octocat@github.com"

inputs:

MY\_NAME:

description: "Who to greet"

required: true

default: "World"

runs:

using: "docker"

image: "Dockerfile"

branding:

icon: "mic"

color: "purple"

**Next, we'll define a workflow that uses the GitHub Action. Workflow File.** A **workflow** file can be thought of as the recipe for automating a task. They house the start-to-finish instructions, in the form of jobs and steps, for what should happen based on specific triggers.

Your repository can contain multiple **workflow** files that carry out a wide variety of tasks. It is important to consider this when deciding on a name for your **workflow**. The name you choose should reflect the tasks being performed.

Workflows are defined in special files in the

.github/workflows directory, named main.yml.

Workflows can execute based on your chosen event. For this lab, we'll be using the [push](https://developer.github.com/v3/activity/events/types/#pushevent) event.

Create a file titled .github/workflows/main.yml

name: A workflow for my Hello World file

on: push

Nice work! 🎉 You added a workflow!

Here's what it means:

* name: A workflow for my Hello World file gives your workflow a name. This name appears on any pull request or in the Actions tab. The name is especially useful when there are multiple workflows in your repository.
* on: push indicates that your workflow will execute anytime code is pushed to your repository, using the [push](https://developer.github.com/v3/activity/events/types/#pushevent) event.

Next, we need to specify a job or jobs to run.

### Actions

Workflows piece together jobs, and jobs piece together steps. We'll now create a job that runs an action. Actions can be used from within the same repository, from any other public repository, or from a published Docker container image. We'll use an action that we'll define in this repository.

Nice, you just added an action block to your workflow file! Here are some important details about why each part of the block exists and what each part does.

* jobs: is the base component of a workflow run
* build: is the identifier we're attaching to this job
* name: is the name of the job, this is displayed on GitHub when the workflow is running
* runs-on: defines the type of machine to run the job on. The machine can be either a GitHub-hosted runner or a self-hosted runner.
* steps: the linear sequence of operations that make up a job
* uses: actions/checkout@v1 uses a community action called [checkout](https://github.com/actions/checkout) to allow the workflow to access the contents of the repository
* uses: ./action-a provides the relative path to the action we created in the action-a directory of the repository
* with: is used to specify the input variables that will be available to your action in the runtime environment. In this case, the input variable is MY\_NAME, and it is currently initialized to "Mona".

### Your action has been triggered!

Your repository now contains an action (defined in the /action-a/ folder) and a workflow (defined in the ./github/workflows/main.yml file).

This action will run any time a new commit is created or pushed to the remote repository. Since you just created a commit, the workflow should have been triggered. This might take a few minutes since it's the first time running in this repository.

As a final step, merge this pull request so the action will be a part of the main branch.

Anyone that uses this repository, and any future code can benefit from this workflow and your new action!