**Practical Exercise 06 - Working with Different Contexts**

**Exercise Description**

**In this practical exercise, our goal is to work with different contexts available during workflow runs.**

Here are the instructions for the exercise:

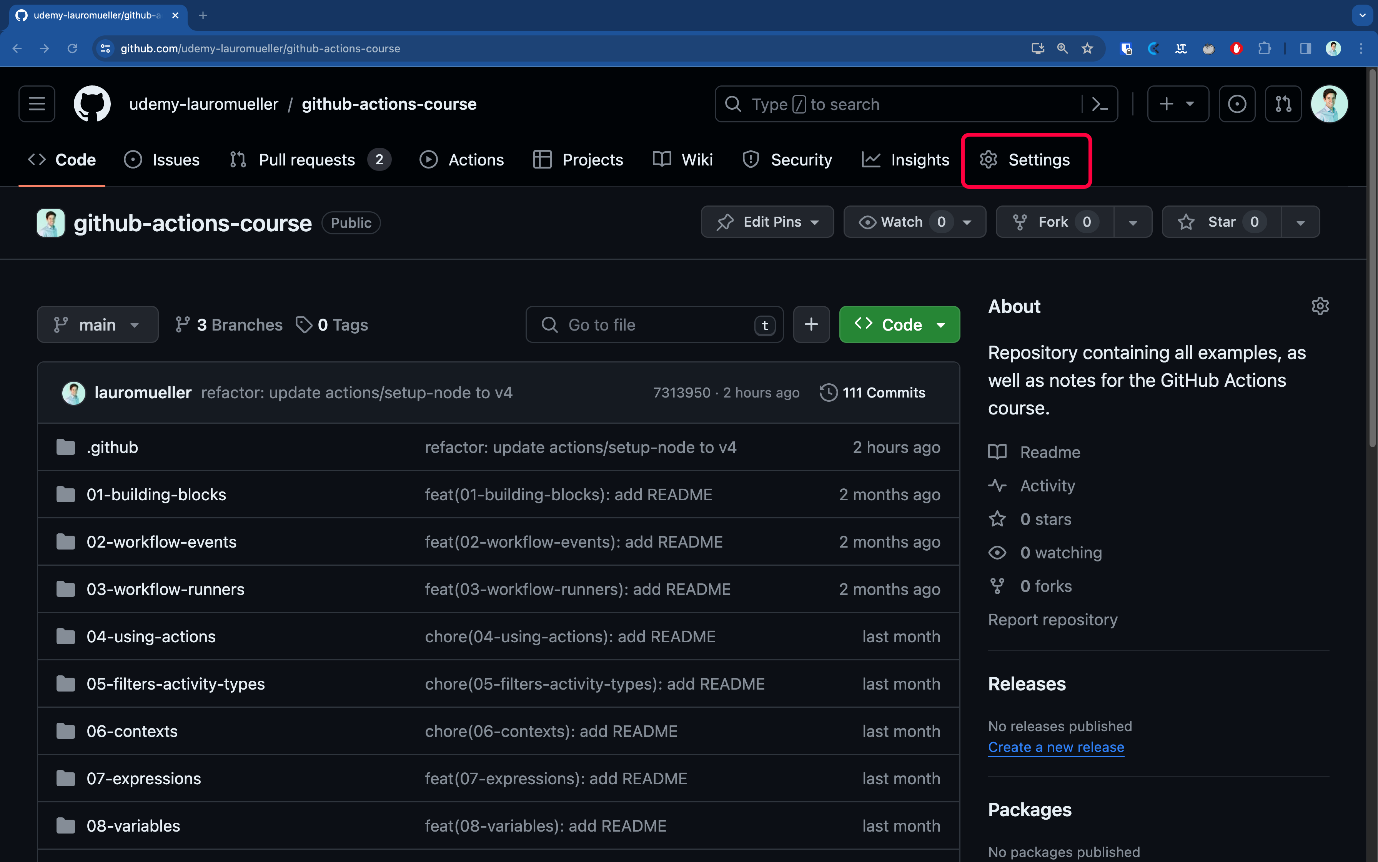
1. Creating the first version of the workflow:
   1. Create a file named 06-contexts.yaml under the .github/workflows folder in the root of your repository.
   2. Name the workflow 06 - Contexts.
   3. Add the following triggers to your workflow:
      1. push
      2. workflow\_dispatch
   4. Add a single job to the workflow. The job, named echo-data, should run on ubuntu-latest and contain two steps:
      1. The first step, named Display Information, should print the following lines containing information from the github context (tip: use a multi-line script with several echo commands):
         1. "Event name: <retrieve the event name here>"
         2. "Ref: <retrieve the ref here>"
         3. "SHA: <retrieve the commit sha here>"
         4. "Actor: <retrieve the actor name here>"
         5. "Workflow: <retrieve the workflow name here>"
         6. "Run ID: <retrieve the run ID here>"
         7. "Run number: <retrieve the run number here>"
      2. The second step, named Retrieve Variable, should print a single line containing the value of a repository variable named MY\_VAR.
   5. Create a repository variable named MY\_VAR. Check the **Tips** section below for the step-by-step of how to create repository variables. Set the value of this variable to hello world.
   6. Commit the changes and push the code. Take some time to inspect the output of the workflow run.
2. Extending the workflow with invalid contexts:
   1. Now add a new configuration key named run-name to the workflow. This is defined at the top-level, as a sibling to the name and on keys. The run-name allows you to define the name of the workflow run that appears on the UI.
   2. Set the value of run-name to My custom workflow run name - ${{ runner.os }}
   3. Commit the changes and push the code. Take some time to inspect the output of the workflow run. Which error message appeared?
3. Fixing invalid contexts and extending the workflow with inputs:
   1. Replace the run-name with 06 - Contexts | DEBUG - ${{ inputs.debug }}
   2. Add an input to the workflow\_dispatch trigger. The input should be named debug, have a boolean type, and have a default value of false. If you are not sure how to do it, check the **Tips** section below for the step-by-step of how to define inputs for the workflow\_dispatch trigger.
   3. Commit the changes and push the code. Take some time to inspect the output of the workflow run triggered by the push event trigger. Which value was populated for the debug input?
   4. Now trigger the workflow from the UI and try it with different variations for the debug input. How does this impact the result of the workflow runs?
4. Extending the workflow with the env context:
   1. Add a top-level env key to define two environment variables:
      1. MY\_WORKFLOW\_VAR, with the value set to 'workflow'
      2. MY\_OVERWRITTEN\_VAR, with the value set to 'workflow'
   2. Under the echo-data job, add an env key to define two environment variables:
      1. MY\_JOB\_VAR, with the value set to 'job'
      2. MY\_OVERWRITTEN\_VAR, with the value set to 'job'
   3. Add an additional step after the Retrieve Variable step, and name it Print Env Variables.
   4. Under the Print Env Variables, add an env key to define a single environment variable:
      1. MY\_OVERWRITTEN\_VAR, with the value set to 'step'
   5. Execute a multi-line script to print the following information on the screen:
      1. "Workflow env: <retrieve the value of the MY\_WORKFLOW\_VAR env variable here>"
      2. "Overwritten env: <retrieve the value of the MY\_OVERWRITTEN\_VAR env variable here>"
   6. Add yet another step after the first Print Env Variables. You can name the step similarly to the previous one. Do not define any additional env variables.
   7. Execute a multi-line script to print the following information on the screen:
      1. "Workflow env: <retrieve the value of the MY\_WORKFLOW\_VAR env variable here>"
      2. "Overwritten env: <retrieve the value of the MY\_OVERWRITTEN\_VAR env variable here>"
   8. Commit the changes and push the code. Take some time to inspect the output of the workflow run. How were the env variables overwritten regarding the workflow, job, and step hierarchy?
5. After exploring the different ways to trigger a workflow, reduce the list of triggers to leave only workflow\_dispatch to prevent this workflow from running with every push and pollute the list of workflow runs.

**Tips**

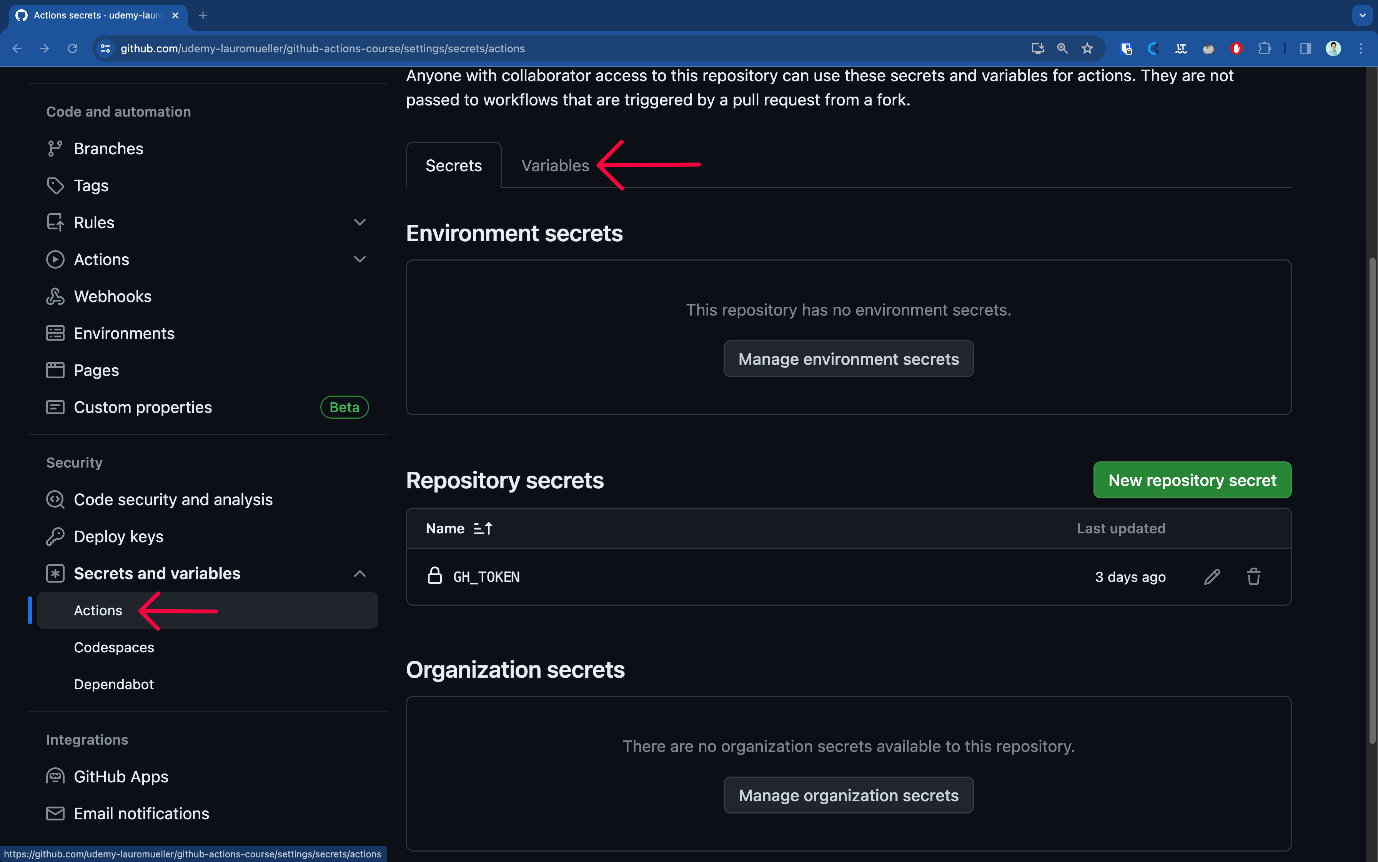
**Creating repository variables**

To create repository variables, follow these steps:

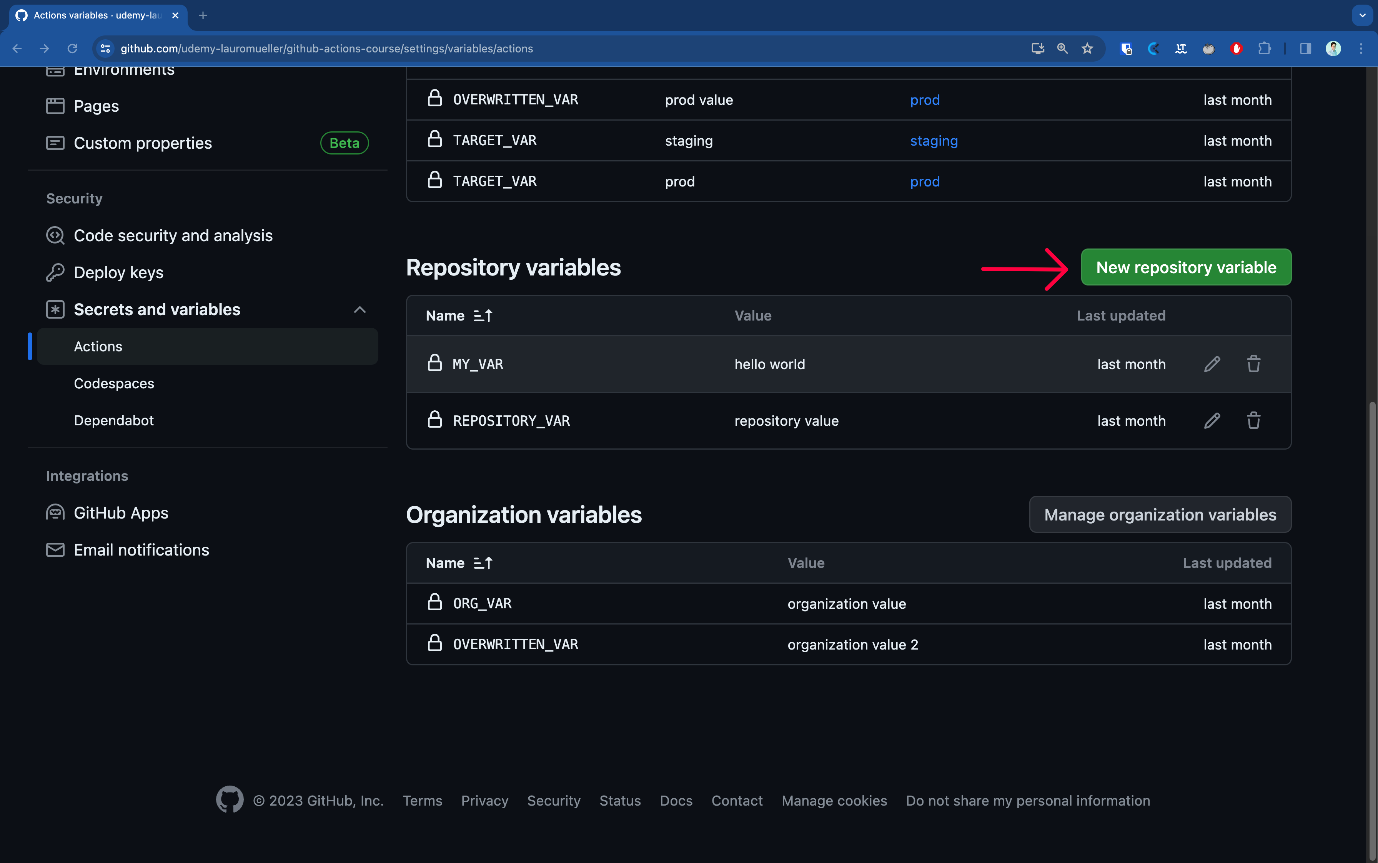
1. In the repository page, click on Settings at the right top side of the screen.



2. Scroll down and expand the Secrets and variables section on the left-side menu. Click on Actions, and then click on Variables on the tabs visible under the description text.



3. Scroll down and click on the New repository variable button.



**Defining inputs for the workflow\_dispatch trigger**

We discuss inputs and outputs in details at **Section 12 - Inputs and Outputs**, but we can already dip our toes in water and define some inputs for the workflow\_dispatch trigger. The syntax to define an input is as follows:

1. on:
2. workflow\_dispatch:
3. inputs:
4. debug:
5. type: boolean
6. default: false

This will add a checkbox to our UI pop up when triggering the workflow.