

In-Class Lab – Continuous Integration
Due: December 2nd, 2022

Today we are going to setup GitHub to handle continuous integration testing. You may have heard of other similar services (e.g., Travis CI, Jenkins, etc.), however those are paid services and I don't want you putting your credit card in.

Basically, you will create a repository and every time you perform `git push` a set of defined tests will run automatically.

For this lab, we'll be using a **new personal repository** so that we don't accidentally break something in your term projects.

Onwards!

First and foremost, create a new repository in your personal GitHub account. Call it `CIS350-<yourlastname>-CI-Lab`.







CONFIGURATION AND TESTING TIME.

I'm writing this from the context of needing to use the EOS server to ensure everybody has access to git and Python.

Go to: <https://cislab.hpc.gvsu.edu/>

I'm going to login to the EOS server – it *should* work on Arch as well but I did not fully test it (so YMMV). Click EOS RDP (or SSH if you're comfortable with the terminal). I will be writing this from the context of a desktop environment.

ALL CONNECTIONS

-  ARCH RDP
-  ARCH SSH
-  DataComm RDP
-  DataComm SSH
-  EOS RDP
-  EOS SSH

Then, login with your GVSU username and password. You'll then be dumped to a Linux desktop. We'll be using a Linux terminal for all of our Git needs, but the file manager should be helpful.

First, make sure your repo is cloned and up to date on your computer. Open up a terminal and run:

```
git clone <repo-http-address>
```

Then, change your directory to that folder:

```
cd CIS350-<yourlastname>-CI-Lab
```

(Note: you can hit Tab to autocomplete the folder name once you start typing – gives you the ability to fly on the terminal).

In your local repository, create two files:

- `math_functions.py`
- `test_math_functions.py`

You can do this however you want (there is PyCharm and Visual Studio installed, but I believe you need to activate them), but if you're unfamiliar with Linux you might want to open the Text Editor application. Click the Activities menu and then the 9-dot menu and search for Text Editor. You can use it like Notepad on Windows.

One thing to note, Python is indentation-sensitive. If you want an if statement, typically you'd write it like this:

```
if a == True:
    print("This was true")
```

And you need to follow that indentation as you go.

Ok, let's add some code. In `math_functions.py` add the following:

```
def add_numbers(a, b):
    return a + b

def subtract_numbers(a, b):
    return a - b

if __name__ == "__main__":
    print("Adding:", add_numbers(2,4))
    print("Subtracting:", subtract_numbers(9,2))
```

You can test it by running:

```
python3 math_functions.py
```

Simple enough. Add two more functions for multiplying and dividing (def is how you create a function in Python).

b) Develop unit tests

Now, open up `test_math_functions.py`. We're going to add some basic unit tests that can be run with pytest.

First, we need to import the functions from `math_functions.py` and then write our test cases.

```
from math_functions import *

def test_calc_addition():
    output = add_numbers(2,4)
    assert output == 6

def test_calc_substraction():
    output = subtract_numbers(2, 4)
    assert output == -2

def test_calc_multiply():
    output = multiply_numbers(2,4)
    assert output == 8

def test_calc_multiply_fail():
    output = multiply_numbers(2,4)
    assert output == 16

def test_calc_divide():
    output = divide_numbers(10,2)
    assert output == 5
```

You can run pytest using the following (it will automatically look for files that start with `test_` to run):

```
python3 -m pytest
```

If you implemented the multiply and divide functions in the same style as the other functions then you should only see a single failing test. If you see other errors read through them to see what is wrong.

If you comment out the `test_calc_multiply_fail` function (i.e., put a `#` in front of each line) and you run pytest again, you should see that all tests pass.

Take a screenshot of your tests passing and paste it into Q1

Ok, local testing done, on to continuous integration!

If you need additional help, here is a reference for pytest: <https://www.softwaretestinghelp.com/pytest-tutorial/>

Continuous Integration with GitHub Actions

Now, go to your GitHub repository and click the 'Actions' tab. Search for Python in the box and select 'Python Application.' Click Configure and off we go.

Get started with GitHub Actions

Build, test, and deploy your code. Make code reviews, branch management, and issue triaging work the way you want. Select a workflow to get started.

Skip this and [set up a workflow yourself](#) →

Categories

- Automation
- Continuous integration
- Deployment
- Security
- Pages

Found 30 workflows

- Django**
By GitHub Actions
Build and Test a Django Project
Configure Python
- SLSA Generic generator**
By Open Source Security Foundation (OpenSSF)
Generate SLSA3 provenance for your existing release workflows
Configure Go
- Pylint**
By GitHub Actions
Lint a Python application with pylint.
Configure Python
- Python application**
By GitHub Actions
Create and test a Python application.
Configure Python
- Python Package using Anaconda**
By GitHub Actions
Create and test a Python package on multiple Python versions using Anaconda for package management.
Configure Python
- Python package**
By GitHub Actions
Create and test a Python package on multiple Python versions.
Configure Python
- Publish Python Package**
By GitHub Actions
Configure Python
- Deploy a Python app to an Azure Web App**
Configure Python
- CodeQL Analysis**
By GitHub

Take a moment to read through the YAML file that was created for you to see what it does. The summary is that it will run any code you push on an Ubuntu image using whichever version(s) of Python you specify. You can configure this to run on multiple architectures and multiple Python versions (or any language you pick) simultaneously, but we'll leave the defaults for now. You should also note that the last lines will be running pytest, which is the focus of our lab here. We can do so much more if we want, but for now let's be satisfied with automated unit tests.

Click the big green 'Start Commit' button and then 'Commit New File.' Congrats, you have setup automated continuous testing on your GitHub repo. Now, whenever a Python file is pushed it will automatically run a basic testing cycle. Note: you may get an automated email saying your test has failed – this makes sense as you haven't pushed any code yet.

A diversion – SSH push

We might as well set up SSH push as managing personal access tokens is mildly annoying at best. In Linux this is pretty straightforward.

(This is the link I have bookmarked whenever I forget how to do this:
<https://gist.github.com/xirixiz/b6b0c6f4917ce17a90e00f9b60566278>)

In your Linux terminal, generate an SSH key pair:

```
ssh-keygen -t rsa -b 4096 -C "your_email@example.com"
```

Hit enter to accept the default location. Set a password if you want as well – this will need to be something you remember for when you go to push. You can also just hit enter to leave it blank.

None of the copy/paste packages are installed in EOS (as far as I'm aware) so we'll need to do this the old fashioned way.

In the terminal, run the following command:

```
cat ~/.ssh/id_rsa.pub
```

This should output the contents of your public SSH key. Select the output, right click and copy (Ctrl+C doesn't always work in this view), and then open up <https://github.com/settings/keys> in a new tab.

Click the New SSH Key button and paste your key into the big Key box. Give it a nice title as well so you remember.

SSH keys / Add new

Title

Key type

Authentication Key ▾

Key

Begins with 'ssh-rsa', 'ecdsa-sha2-nistp256', 'ecdsa-sha2-nistp384', 'ecdsa-sha2-nistp521', 'ssh-ed25519', 'sk-ecdsa-sha2-nistp256@openssh.com', or 'sk-ssh-ed25519@openssh.com'

Add SSH key

Almost ready. Now, run this command to associate your account with GitHub (and test to make sure it works).

```
ssh -T git@github.com
```

Last but not least, in your repository run the following to associate your repository with SSH push access.

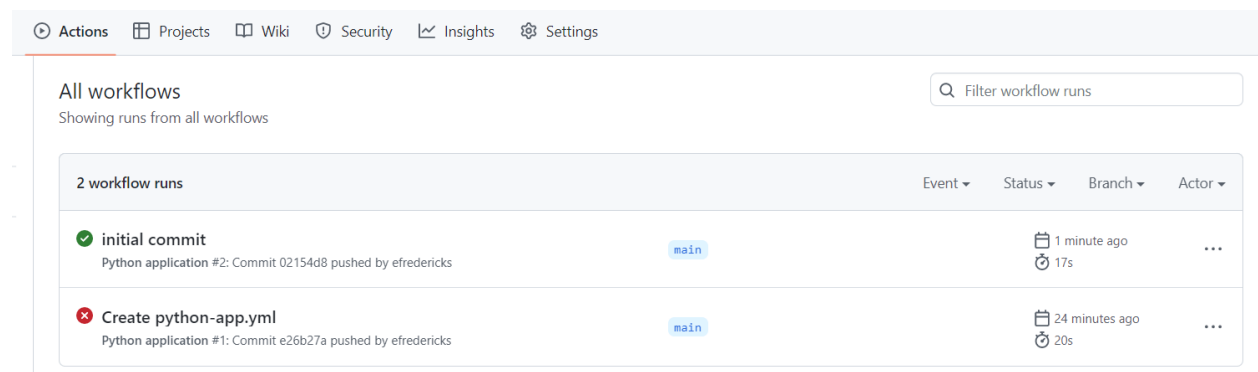
```
git remote set-url origin git@github.com:<your GitHub username>/<your GitHub repository>
```

Ok, now we should be able to use Git as normal. Except when you push now, it will ask you for the password of your SSH key you set above (if you set a password).

```
git add .
git commit -m 'initial commit'
git push
```

Generating your Test Report

If everything went well with your commit and your push, you should have automatically kicked off your Action workflow. Click the Actions tab again and look at your most recent commit. If your tests were passing locally, you should see a big green checkmark that states you passed. If you didn't fix your tests, you'll notice it failed.



The screenshot shows the GitHub Actions interface. At the top, there's a navigation bar with tabs: Actions, Projects, Wiki, Security, Insights, and Settings. Below this, the 'All workflows' section is active, showing 'Showing runs from all workflows'. A search bar labeled 'Filter workflow runs' is present. Below the search bar, a table titled '2 workflow runs' displays the following data:

Event	Status	Branch	Actor
initial commit Python application #2: Commit 02154d8 pushed by efredericks	Success (Green checkmark)	main	1 minute ago 17s
Create python-app.yml Python application #1: Commit e26b27a pushed by efredericks	Failure (Red X)	main	24 minutes ago 20s

Now, uncomment your 'failing' multiplication test to force it to fail. Run your add/commit/push commands again to generate a failing test.

You should be able to click the failed action, then 'build' on the side to drill down into what went wrong.

Summary

Jobs

build

Run details

Usage

Workflow file

build

failed 2 minutes ago in 6s

Lint with flake8

Test with pytest

```
1 ▶ Run pytest
7 ===== test session starts =====
8 platform linux -- Python 3.10.8, pytest-7.2.0, pluggy-1.0.0
9 rootdir: /home/runner/work/pyc12/pyc12
10 collected 5 items
11
12 test_math_functions.py ...F. [100%]
13
14 ===== FAILURES =====
15 _____ test_calc_multiply_fail _____
16
17     def test_calc_multiply_fail():
18         output = multiply_numbers(2,4)
19     > assert output == 16
20 E       assert 8 == 16
21
22 test_math_functions.py:17: AssertionError
23 ===== short test summary info =====
24 FAILED test_math_functions.py::test_calc_multiply_fail - assert 8 == 16
25 ===== 1 failed, 4 passed in 0.03s =====
26 Error: Process completed with exit code 1.
```

Post Set up Python 3.10

Post Run actions/checkout@v3

Complete job

Congrats, you now have a test runner with probably a lot of automated emails. **Fix your broken test case, push again, and take a screenshot of it passing and paste into Q2.**

If you need additional help, here is an abbreviated reference for GitHub Actions + CI:

<https://python.plainenglish.io/setting-up-ci-in-your-python-project-using-github-actions-devops-learning-edition-8e496503f89c>

Lab Report

Q1: Screenshot of local unit tests passing.

```
• jwill@LAPTOP-990487PQ:~/GVSU/CIS-350/CIS350-Willey-CI-Lab$ python3 -m pytest
===== test session starts =====
platform linux -- Python 3.10.7, pytest-7.2.0, pluggy-1.0.0
rootdir: /home/jwill/GVSU/CIS-350/CIS350-Willey-CI-Lab
collected 4 items

test_math_functions.py .... [100%]

===== 4 passed in 0.01s =====
```

Q2: Screenshot of GitHub actions tests passing.

✓ Create python-app.yml
Python application #1: Commit d7abc46 pushed by Jayson729
main
23 minutes ago
20s

Q3: Add two new mathematical functions and two new unit tests in the Test-Driven Development style (i.e., add tests first, then add the code). **Show the local results** of your tests failing (because only your unit tests exist) then passing (because you added the code to make them pass).

```
jwill@LAPTOP-990487PQ:~/GVSU/CIS-350/CIS350-Willey-CI-Lab$ pytest
===== test session starts =====
platform linux -- Python 3.10.7, pytest-7.2.0, pluggy-1.0.0
rootdir: /home/jwill/gvsu/cis-350/cis350-willey-ci-lab
collected 6 items

test_math_functions.py ....FF [100%]

===== FAILURES =====
_____ test_calc_exponent _____

    def test_calc_exponent():
        output = exponentiate_numbers(5, 3)
>       assert output == 125
E       assert None == 125

test_math_functions.py:31: AssertionError
_____ test_calc_mod _____

    def test_calc_mod():
        output = mod_numbers(15, 4)
>       assert output == 3
E       assert None == 3

test_math_functions.py:36: AssertionError
===== short test summary info =====
FAILED test_math_functions.py::test_calc_exponent - assert None == 125
FAILED test_math_functions.py::test_calc_mod - assert None == 3
===== 2 failed, 4 passed in 0.10s =====

jwill@LAPTOP-990487PQ:~/GVSU/CIS-350/CIS350-Willey-CI-Lab$ pytest
===== test session starts =====
platform linux -- Python 3.10.7, pytest-7.2.0, pluggy-1.0.0
rootdir: /home/jwill/gvsu/cis-350/cis350-willey-ci-lab
collected 6 items

test_math_functions.py ..... [100%]

===== 6 passed in 0.02s =====
```

Q4: Commit them to your GitHub repository. Show a passing GitHub Actions report that includes your new functions.

```
✓ Merge branch 'main' of github.com:Jayson729/CIS350... 4 minutes ago
Python application #2: Commit 7f6c688 pushed by Jayson729 23s
```

Q5: Paste a link to each file (the unit test and main function) from your GitHub repository here.

Repo: <https://github.com/Jayson729/CIS350-Willey-CI-Lab>

Main: https://github.com/Jayson729/CIS350-Willey-CI-Lab/blob/main/math_functions.py

Test: https://github.com/Jayson729/CIS350-Willey-CI-Lab/blob/main/test_math_functions.py

Q6: Want extra credit? Add coverage testing to your CI chain to generate a coverage report. Describe (1) what you did and (2) provide screenshots of it in action. Note: you'll most likely need to turn your math_functions file into a class. If you want extra-extra credit ensure you have 100% coverage!