# COMP3006 – Full-Stack Development

## Workshop 11 – Continuous Integration

Autumn 2020

During this week's lecture you learned about **continuous integration**. In this week's lab you will what you've learned so far into practice and use continuous integration for one of the example projects on the DLE

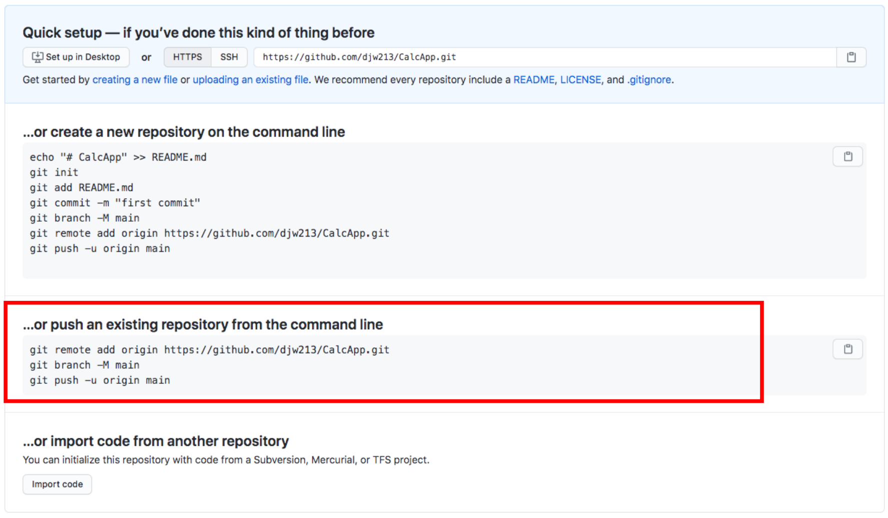
You will also extend your work on the mini project by setting up continuous integration for the code you have been writing in recent weeks.

### Exercise 1

Begin by downloading the demo code Zip from Week 10 (last week's lecture demos, in which I showed you how to write unit and integration tests for the calculator application). In the terminal, **change directory** into the folder you've unzipped and use the following Git commands to initialise the repo:

```
$ git init
$ git add *
$ git commit -m "Initial commit of calculator project."
```

Then, go to GitHub and create a new project (call it **CalcApp**). You will see the following three options – you want the middle one (push an existing repo from the command line):



**Quick setup — if you've done this kind of thing before**

Set up in Desktop or HTTPS SSH `https://github.com/djw213/CalcApp.git`

Get started by creating a new file or uploading an existing file. We recommend every repository include a README, LICENSE, and .gitignore.

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

```
echo "# CalcApp" >> README.md
git init
git add README.md
git commit -m "first commit"
git branch -M main
git remote add origin https://github.com/djw213/CalcApp.git
git push -u origin main
```

**...or push an existing repository from the command line**

```
git remote add origin https://github.com/djw213/CalcApp.git
git branch -M main
git push -u origin main
```

**...or import code from another repository**

You can initialize this repository with code from a Subversion, Mercurial, or TFS project.

Once you've done that you should have a repo with the code in your GitHub profile.

### Exercise 2

If you don't have one, start an account with **Travis CI** and link it to your GitHub profile.

## Exercise 3

Use the following terminal command to initialise your Node package:

```
$ npm init
```

Follow the prompts to complete the `package.json` file as we did in Lecture 6. The entry point should be **server.js**, and the test script should be `mocha -ui tdd test/`. Then use

```
$ git add package.json
$ git commit -a -m "Adding package.json"
```

to add the new file to the repo and commit it.
Add a `.travis.yml` and modify it as follows:

Please note the preceding dot of `.travis.yml`!

```
language: node_js
node_js:
- node
install:
- npm install -g mocha
- npm install chai
- npm install chai-http
- npm install express
```

Once you've done that, add the new file to the repo, commit and push as follows:

```
$ git add .travis.yml
$ git commit -a -m "Adding .travis.yml"
$ git push
```

Your test should now pass. Check your Travis build history to ensure that it does.

## Exercise 4

Once you've got the CI pipeline working, you can start to add to the code. Modify the **logic.js** file to add a function called **addNumbers** which takes two variables and returns the result of adding them together.

You did all of this last week – have a look at the lecture and workshop model answers to refresh yourself if necessary.

Having implemented the function, add a unit test suite to **unit.js** that tests that the function operates correctly.

## Exercise 5

Extend the previous exercise by exposing the new function through the API:

You did all of this last week too – have a look at the lecture and workshop model answers to refresh yourself here too if necessary.

- Modify **routes.js** to include a route that unpacks two parameters (**number1** and **number2**) and uses them to call the **addNumbers** function, returning the result.
- Add an **app.get** rule to call invoke the route when the correct URL pattern is used. Remember to include the parameters.

Remember to convert your numbers to **Number** form before sending to the **addNumbers** function, and to convert its result to a **String** with **toString** before returning with **response.send**.

Add an integration test to ensure that the new route works properly.

## Mini Project

This week you will extend the mini project by adding what you have learned this week about continuous integration.

### Exercise 6

Last week you unit tested and integration tested your application's server side. This week you should set up your project to use continuous integration.

## Extension task

### Exercise 7

In the exercises above you set up last week's demo code for continuous integration. Have a go at deploying with continuous deployment, so that when your Travis build is complete it deploys to Heroku. You will need to research how to do this online.