# Assignment: Continuous Integration with GitLab

## Overview

In this assignment, you will gain hands-on experience with Continuous Integration (CI) by setting up a pipeline on GitLab to run a unit test and two security scans on a JavaScript program. This activity will introduce you to key CI concepts, such as pipelines, stages, and jobs, and provide a foundation for understanding how CI improves software quality.

## Learning Objectives

* Understand the purpose of CI in software development.
* Set up and trigger a basic CI pipeline on GitLab.
* Write and run unit tests using Jest.
* Perform two security scans:  
   - A JSHint scan to identify potential errors and enforce coding standards.  
   - A Secret Detection scan to look for sensitive information.
* Use Git to push code to a GitLab repository.
* Reflect on the role of CI in collaborative development.

## Part 1: Develop Your JavaScript Programs

1. Create a project directory, which will contain all of your code and library installs, on your machine.
2. Create a program (validatePalindrome.js) that:
   1. Accept an array of strings as input.
   2. Checks if the array element is a palindrome
   3. The check should ignore case, periods, commas and whitespace
   4. Array elements that are palindromes should be written to the console.
3. The program should conform to the following design standards:
   1. Uses a function named cleanData(inputWord) that:
      1. Converts all letters in the string to lowercase.
      2. Removes all spaces, commas, and periods.
      3. Returns the cleaned string.
   2. Has a function named testPalindrome(cleanedWord) that:
      1. Reverses the cleaned string.
      2. Compares it to the original cleaned string.
      3. Returns true if they match, false otherwise.
   3. Has a function named validatePalindrome(inputWord) that:
      1. Calls cleanData() to clean the input.
      2. Calls testPalindrome() to check if the cleaned string is a palindrome.
      3. Returns true or false.
4. Example input and output
   1. Input: ["racecar", "hello", "Level", "I", "world", "mad am", "12321", "not a palindrome"]
   2. Output: racecar, Level, I, mad am, 12321
5. After your program runs successfully on your laptop: Prepare your program for the CI/CD pipeline

* At the top of the file, add this comment to specify the JavaScript version for the linter (JSHint):

/\* jshint esversion: 6 \*/

* At the bottom of your program, add:

module.exports = validatePalindrome;

* Remove your input array and console.log() statements
* Do not call the main() program. You will test your function using Jest in the step.

1. Write unit tests using Jest:
   1. Create a file called validatepalindrome.test.js. This file will contain your unit tests for the validatePalindrome function.
   2. A working example is posted on Canvas. The filename is:  
      example\_test\_jest.js.
   3. Use the example to define **at least three unit test cases**.  
      For example, test that the string "hello" is **not** identified as a palindrome.
   4. Your tests should call validatePalindrome() and use expect() to check the result. Each test should clearly describe what it's checking.

## Part 2: Set up your GitLab project

1. Create a GitLab Repository: Go to <https://gitlab.com> and sign in (or create an account).
2. Create a new group (if one doesn’t exist):
   * 1. Go to <https://gitlab.com/groups>
     2. Click "New group"
     3. Group name: INFO 465 (or your preferred name)
     4. Visibility level: Private
3. Create a new project
   * 1. Choose the Blank Project template
     2. Project name: pipeline-build-assignment
     3. Leave README unchecked
4. Clone the Repository to your computer. This creates the working folder you’ll use from now on.
   * 1. Click the blue “Clone” button in GitLab
     2. Copy the HTTPS URL, e.g., https://gitlab.com/yourname/pipeline-build-assignment.git
     3. Open a terminal or command prompt, and enter the following commands:
        1. git clone https://gitlab.com/yourname/pipeline-build-assignment.git
        2. cd pipeline-build-assignment
5. Setup Jest in your GitLab project folder on your terminal/command prompt. This within the cloned project folder on your laptop.
   * 1. Type in: npm init -y
     2. Type in: npm install --save-dev jest
     3. Open the file package.json. Find the scripts section and change it to:

"scripts": {

"test": "jest"

}

## Part 3: Configure CI Pipeline

1. Create a .gitlab-ci.yml file. This file defines your pipeline stages and jobs.
   1. In GitLab, go to your project page.
   2. Click **“Add ➕” → “New File”**
   3. Name the file: .gitlab-ci.yml
   4. Cut and paste from the .gitlab-ci.yml file posted on Canvas.
   5. Annotate your yml file by adding a comment, in your words, describing what each step of the job does.
2. Run and monitor your pipeline
   1. Go to CI/CD > Pipelines in your GitLab project. Verify that the pipeline runs and completes successfully.
   2. To view results:
      1. Unit tests: You can view the results of your unit tests in the "test" job’s log output (CI/CD > Pipelines > Job > View Raw)
      2. Code quality (JShint):
         1. Go to the pipeline summary page.
         2. Click the "Browse" button under Artifacts for the jshint\_scan job.
         3. Download or view jshint-report.txt
      3. Secrets:
         1. On the pipeline page, find the job labeled secret-detection (or similar).
         2. Open the job and scroll through the Job Log to see scan output.
         3. If secrets are found, they will be listed in the log or in a downloadable report called gl-secret-detection-report.json under Artifacts.

## Part 4: Submission

Submit the following:

* A screenshot of your pipeline showing a successful run.
* Both JavaScript programs
* Your .yml file
* Download and submit the artifacts from your latest pipeline run (each job’s artifacts).
* A 100–150 word analysis of the scan results.
* A 100–150 word reflection on what you learned and how CI improves collaboration.

## Grading Rubric (100 Points)

- JavaScript Program: 20 points – Fully functional and meets spec. No partial credit.  
- Pipeline Setup & Results: 55 points  
 - 15 points – CI jobs defined correctly in .gitlab-ci.yml. Includes comments/annotation.   
 - 15 points – Successful pipeline run.  
 - 25 points – Thoughtful analysis of scan results and response to added code.  
- Reflection: 20 points  
 - 10 points – Clarity, grammar, and organization.  
 - 10 points – Insightful connection to CI concepts.  
- Clean Submission: 5 points – Clear formatting and clean artifacts.