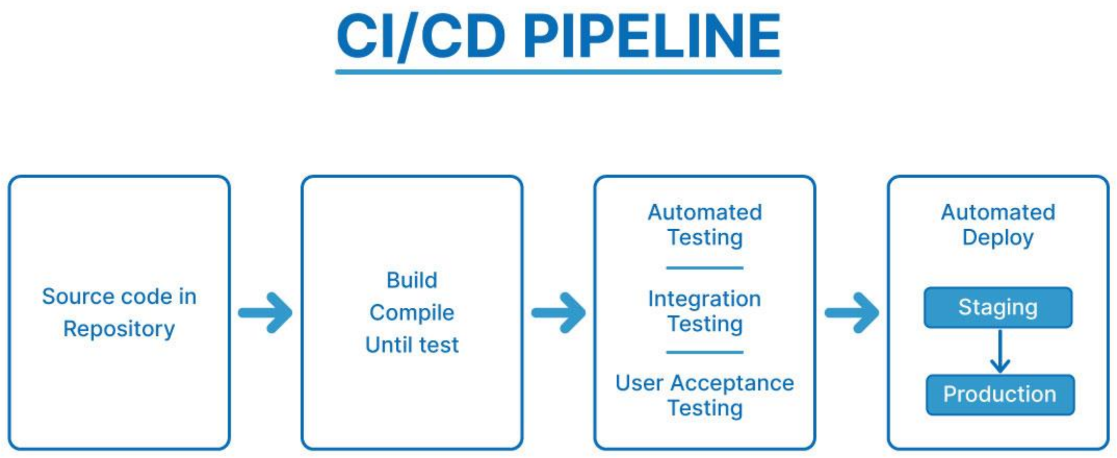
CI/CD PIPELINE

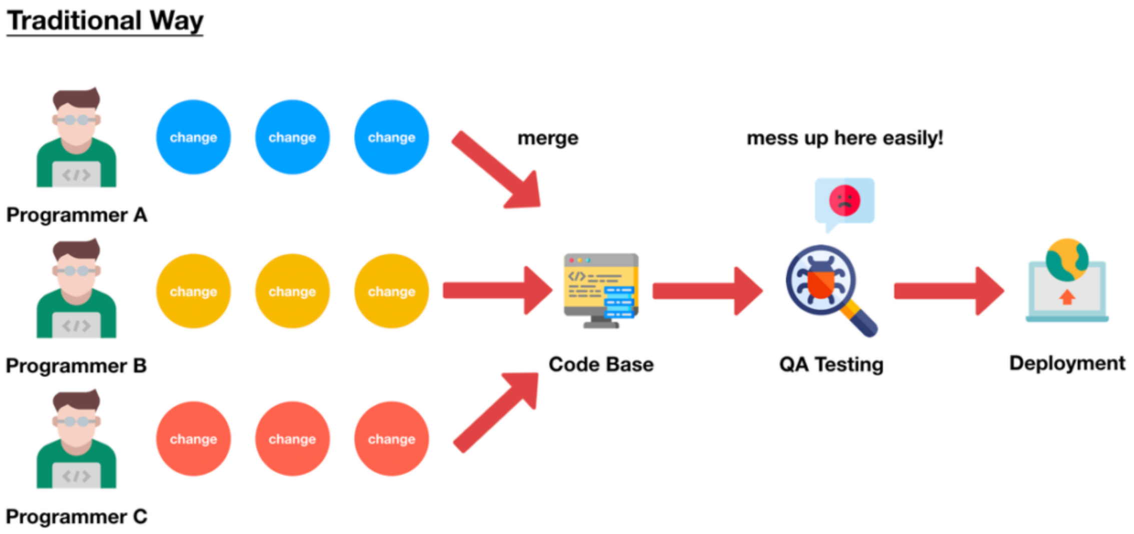
CI/CD stands for Continuous Integration/Continuous Delivery (or Continuous Deployment). It's a set of practices and methodologies used in software development and DevOps to automate the process of integrating code changes into a shared repository (Continuous Integration) and then continuously delivering those changes to production environments .



1. **Continuous Integration (CI)**: Developers regularly merge their code changes into a central repository, where automated builds and tests are run. This ensures that each code change is tested in an isolated environment, helping to catch integration errors early in the development process.

2. **Continuous Delivery (CD)**: This involves automating the deployment process so that any changes that pass the automated tests in the CI stage can be automatically deployed to production or staging environments. Continuous Delivery typically involves deploying changes to a staging environment first, where further testing can be performed before deploying to production.

3. **Continuous Deployment**: This is an extension of Continuous Delivery where every change that passes through the CI pipeline is automatically deployed to production without manual intervention, assuming it passes all tests and quality gates.



Various Tools:

- Jenkins
- GitLab CI/CD
- Travis CI
- CircleCI

GIT and GITHUB

__Git is a distributed version control system (DVCS) used for tracking changes in source code during software development. It allows multiple developers to collaborate on projects and efficiently manage changes to the codebase. Git was created by Linus Torvalds in 2005 and has since become one of the most widely used version control systems in the software development industry.

FEATURES:

>**Version Control**: Git tracks changes to files and directories over time, allowing developers to view, revert, or merge changes as needed

>**Branching and Merging**: Git allows developers to create separate branches to work on features or fixes independently. Branches can be merged back into the main codebase once the changes are complete.

>**Distributed Development**: Git is a distributed version control system, meaning that every developer has a complete copy of the repository, including its entire history. This allows developers to work offline and collaborate with others without relying on a centralized server.

>**Staging Area**: Git has a staging area, also known as the "index," where changes can reviewed and prepared before committing them to the repository.

GitHub :

GitHub, on the other hand, is a web-based platform built around Git that provides hosting for Git repositories. It offers additional features such as issue tracking, project management tools, code review, and collaboration features like pull requests. GitHub allows developers to host their Git repositories remotely and provides a central hub for collaboration on open-source and private projects alike.

Features:

1. **Remote Repository Hosting**: GitHub hosts Git repositories on its servers, allowing developers to push and pull changes to and from remote repositories.

2. **Collaboration Tools**: GitHub provides tools for code review, issue tracking, project management, and collaboration among developers working on the same project.
3. **Pull Requests**: GitHub facilitates code review and contribution through pull requests, which allow developers to propose changes to a project and request that they be reviewed and merged into the main codebase.
4. **Community and Open Source**: GitHub is home to a large community of developers and hosts millions of open-source projects across various domains.

GitAction :

Workflows: A workflow is a configurable automated process that will run one or more jobs. Workflows are defined by a YAML file checked in to your repository and will run when triggered by an event in your repository, or they can be triggered manually, or at a defined schedule.

- Workflows are defined in the `.github/workflows` directory in a repository, and a repository can have multiple workflows, each of which can perform a different set of tasks.

Workflow syntax :

```
jobs:
  setup:
    runs-on: ubuntu-latest
    steps:
      - run: ./setup_server.sh
  build:
    needs: setup
    runs-on: ubuntu-latest
    steps:
      - run: ./build_server.sh
  test:
    needs: build
    runs-on: ubuntu-latest
    steps:
      - run: ./test_server.sh
```

- **`JOBS`** **:** A workflow run is made up of one or more `jobs`, which run in parallel by default. To run jobs sequentially, you can define dependencies
- Each job runs in a runner environment specified by `runs-on`.

Runs-on: Runs-on tells on which operating system we are working on.

Types of runners ->

Self hosted : They can be used for workflows that require specific configurations or access to resources that are not available on GitHub-hosted runners..

- **Github hosted :** These are virtual machines managed by GitHub.
  GitHub provides a variety of pre-configured runners that support different operating systems and software environments, such as Ubuntu, macOS, and Windows.

Matrix : Matrix keyword is used to add different versions of the different versions of dependencies, operating systems, or environments .

Example:

```
matrix:
  Node version:  [10.x,12.x,14.x]
```

Basic commands used to configure :

```
$git init
$ git add .
$ git commit
$ git branch -M main
$ git remote add origin
$ git push
$ git pull
```

Environment variables :

Env keyword is used to set variables at different levels

There are mainly three levels :

1. **Workflow level** :

  In workflow env variables is available for all jobs at all steps with in the workflow.

2. **Job level :**

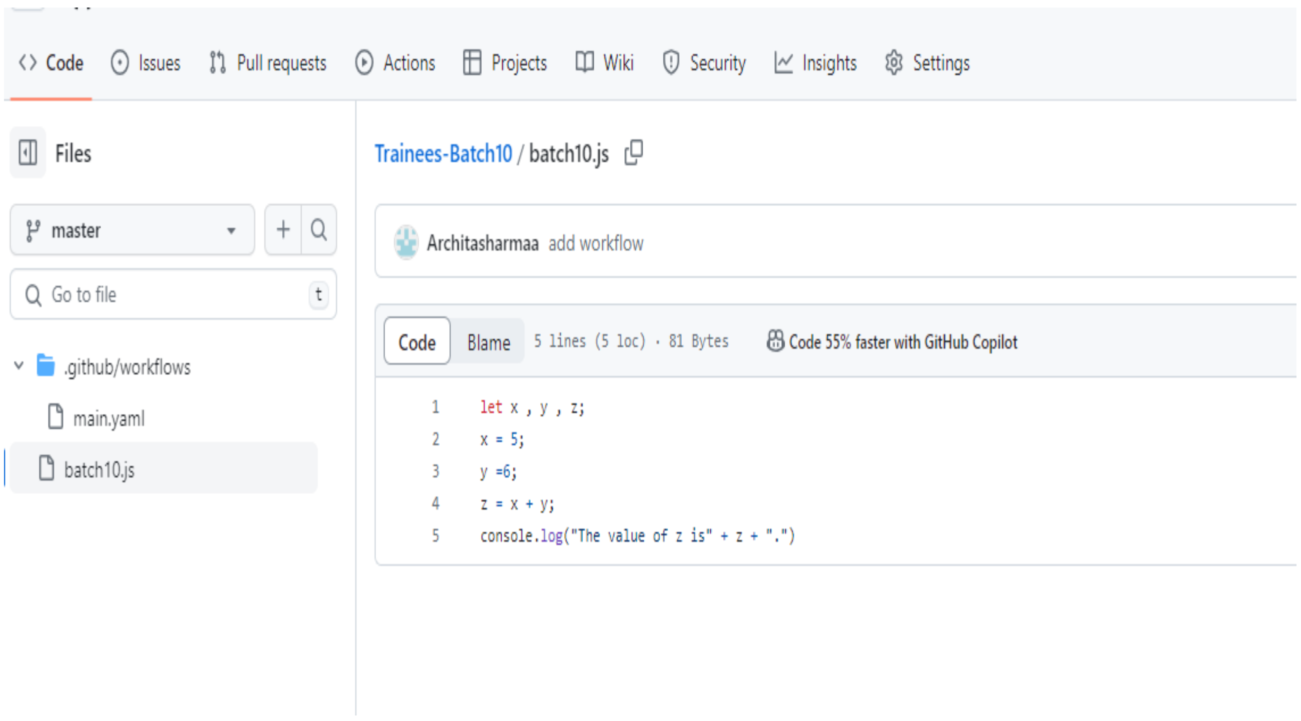
In job level where ever we declare the env variable it is applicable within that jb level

3 Step level :

 When we mention that required variables within step it will applicable or accessed from that steps level only

How to Print

```
Example_Environemnt_Variables:

Runs-on: ubuntu-latest

Env:

First_Variable :”12345”

Steps:

-name :

 Run : echo “first_variable :→ $ {{env.First_Variable}}”
```

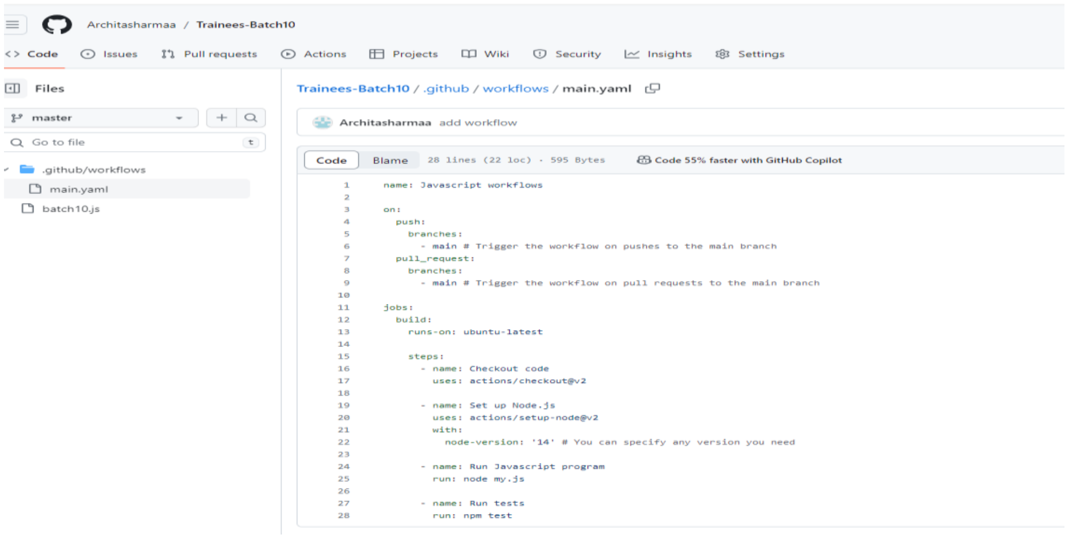


Trainees-Batch10 / batch10.js

Architasharmaa add workflow

5 lines (5 loc) · 81 Bytes

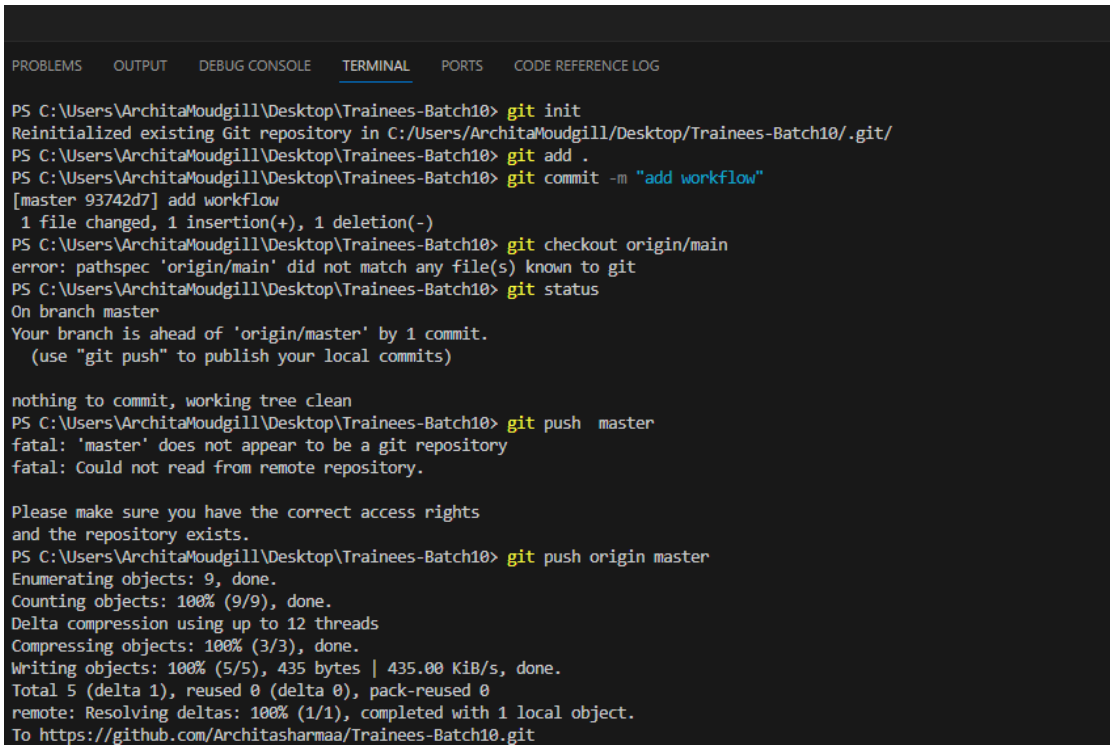
```
let x , y , z;
x = 5;
y =6;
z = x + y;
console.log("The value of z is" + z + ".")
```



Architasharmaa add workflow

28 lines (22 loc) · 595 Bytes

```yaml
name: Javascript workflows

on:
  push:
    branches:
      - main # Trigger the workflow on pushes to the main branch
  pull_request:
    branches:
      - main # Trigger the workflow on pull requests to the main branch

jobs:
  build:
    runs-on: ubuntu-latest

    steps:
      - name: Checkout code
        uses: actions/checkout@v2

      - name: Set up Node.js
        uses: actions/setup-node@v2
        with:
          node-version: '14' # You can specify any version you need

      - name: Run Javascript program
        run: node my.js

      - name: Run tests
        run: npm test
```



```
PS C:\Users\ArchitaMoudgill\Desktop\Trainees-Batch10> git init
Reinitialized existing Git repository in C:/Users/ArchitaMoudgill/Desktop/Trainees-Batch10/.git/
PS C:\Users\ArchitaMoudgill\Desktop\Trainees-Batch10> git add .
PS C:\Users\ArchitaMoudgill\Desktop\Trainees-Batch10> git commit -m "add workflow"
[master 93742d7] add workflow
 1 file changed, 1 insertion(+), 1 deletion(-)
PS C:\Users\ArchitaMoudgill\Desktop\Trainees-Batch10> git checkout origin/main
error: pathspec 'origin/main' did not match any file(s) known to git
PS C:\Users\ArchitaMoudgill\Desktop\Trainees-Batch10> git status
On branch master
Your branch is ahead of 'origin/master' by 1 commit.
  (use "git push" to publish your local commits)

nothing to commit, working tree clean
PS C:\Users\ArchitaMoudgill\Desktop\Trainees-Batch10> git push  master
fatal: 'master' does not appear to be a git repository
fatal: Could not read from remote repository.

Please make sure you have the correct access rights
and the repository exists.
PS C:\Users\ArchitaMoudgill\Desktop\Trainees-Batch10> git push origin master
Enumerating objects: 9, done.
Counting objects: 100% (9/9), done.
Delta compression using up to 12 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (5/5), 435 bytes | 435.00 KiB/s, done.
Total 5 (delta 1), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To https://github.com/Architasharmaa/Trainees-Batch10.git
```