# CICD Implementation

Team 21 Spring 2023

Updated: 6/7/2023

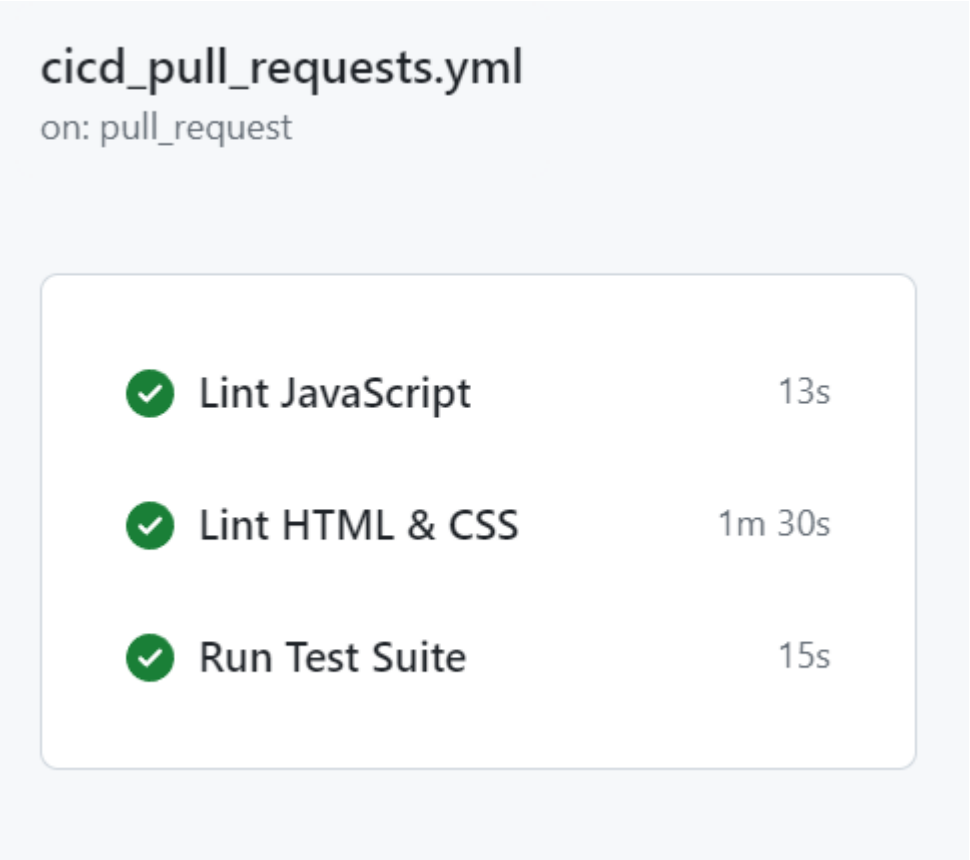
# Branch Structure Strategy

- Two fundamental branches
  - The main branch
    - Stores current stable release*
      - *The main branch can slightly differ in repo-level things (e.g. CI/CD or README changes) – releases are kept safe via tags (e.g., our v0.1.0 is unchanged regardless of what happens to main)
    - Push from dev branch (via pull request) to make a new release
    - Create an offshoot branch from main and make a pull request to merge back into main for repo-level things
      - E.g., CI/CD or README changes
      - May depend on case-by-case basis (for small changes we just want to commit once without a big headache)
  - The dev branch
    - Collective branch for everyone to work on next release
    - Feature / bug-fix branches come from the dev branch and are merged back via pull-request to add new features / bug fixes to development work

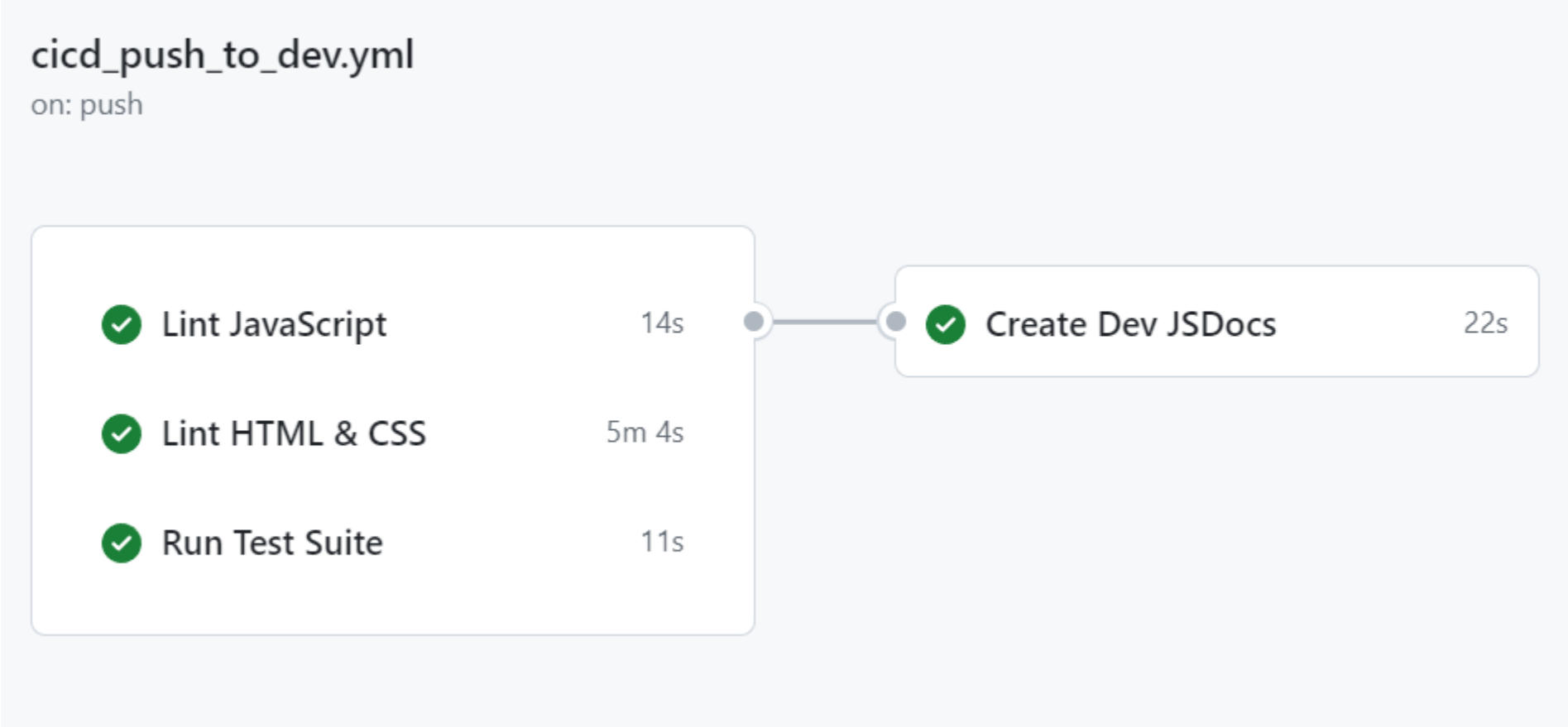
# CI / CD Approach

- Three unique situations:
  - Pushing to the main branch
    - Linting, tests, main branch docs, minimization, deployment
  - Pushing to the dev branch
    - Linting, tests, dev branch docs (stored in a separate folder than main branch docs)
  - Pushing to other branches / creating a pull request
    - Linting and tests
- Each situation has its own workflow (three .yml files)
- Could add more steps / workflows in the future if necessary

# Pushing to other branches / Pull Requests



# Pushing to dev branch



# Pushing to main branch

# Visualized Commit Work

new-feature main dev

**Push to main:** Added fooMain() – triggers push to main workflow

- Workflow creates documentation in the docs/ directory and generates minimized code in the dist/ directory (in two separate commits)

**Push to dev:** Added fooDev() – triggers push to dev workflow

- Workflow creates documentation in the dev-docs/ directory
  - this is done to avoid merge conflicts
  - dev-docs is deleted from the main branch upon merging dev with main to create a new release

**Push to new-feature:** Added fooNewFeature() – triggers pull request or general push workflow

- Lints JavaScript / HTML / CSS and runs test suite, but that's it

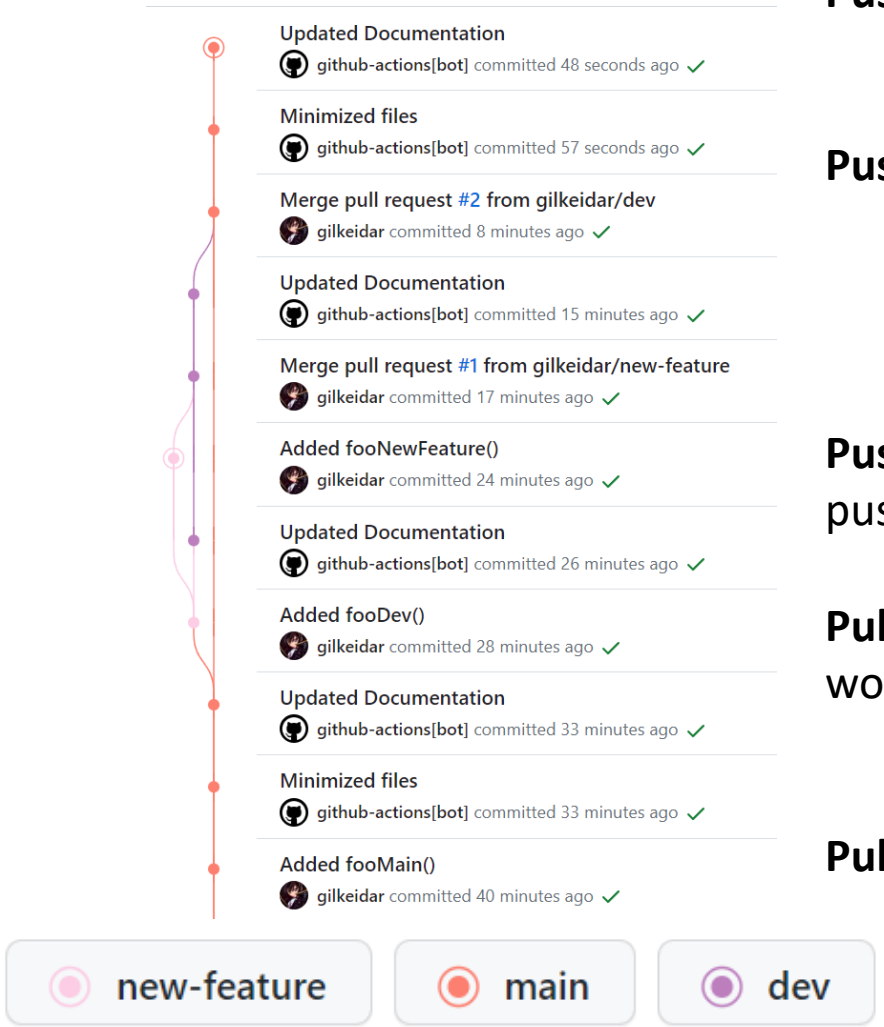
**Pull request from new-feature to dev:** Triggers pull request or general push workflow

- After the pull request is merged to dev, the push to dev workflow will be run since we're pushing to dev

**Pull request from dev to main:** Triggers pull request or general push workflow

- After the pull request is merged to main, the push to main workflow will be run since we're pushing to main

# Visualized Commit Work




**Push to main:** Added fooMain() – triggers push to main workflow

- Workflow creates documentation in the docs/ directory and generates minimized code in the dist/ directory (in two separate commits)

**Push to dev:** Added fooDev() – triggers push to dev workflow

- Workflow creates documentation in the dev-docs/ directory
  - this is done to avoid merge conflicts
  - dev-docs is deleted from the main branch upon merging dev with main to create a new release

**Push to new-feature:** Added fooNewFeature() – triggers pull request or general push workflow

- Lints JavaScript / HTML / CSS and runs test suite, but that's it

**Pull request from new-feature to dev:** Triggers pull request or general push workflow

- After the pull request is merged to dev, the push to dev workflow will be run since we're pushing to dev

**Pull request from dev to main:** Triggers pull request or general push workflow

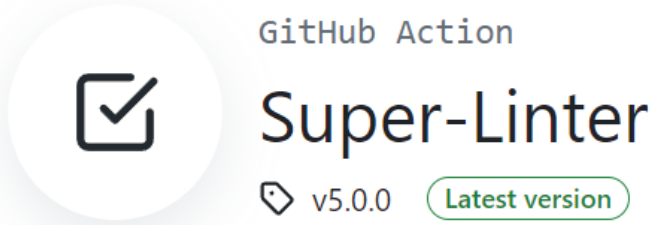
- After the pull request is merged to main, the push to main workflow will be run since we're pushing to main

# CI / CD Current Steps

# Linting - JavaScript


- Linter is used to maintain code style throughout our project
- JavaScript Linter – Semi-Standard Style
  - Allows for semi-colons (important for minifier)
  - Recommended: Install the Standard Style VSCode Extension
    - Found here
    - After installing, go to **Extensions > StandardJS – JavaScript Standard Style > Extension Settings** and set **Standard: Engine** to semistandard.
    - To check that it worked, open a JavaScript file. If the code isn't in Semi-Standard style, you'll see lots of red squiggly underlines. Save the file and it should update the code to conform to this style (you may need to save multiple times)

# Linting – HTML & CSS




- Linter is used to maintain code style throughout our project
- HTML & CSS Linter – Super-Linter
  - Found here
  - Advantages:
    - Easy to use – it's a GitHub Action
    - Supports lots of languages
  - Disadvantages:
    - Slow install time – you can expect it to run for at least 1 minute which slows down the CI/CD pipeline considerably

# Testing - Jest

- Testing JavaScript files with Jest
- Install Jest in the terminal using
  - npm install --save-dev jest
  - Alternatively, the repository's package.json and package-lock.json files already specify the required dependencies, so I believe that you can just pull from the repository and then run npm install instead.
- Put your tests in a tests/ folder outside src/
- Note that at present the JavaScript linter ***does*** lint the tests folder. If this causes issues, it can be easily configured to not lint the tests folder.

# Documentation - JSDocs

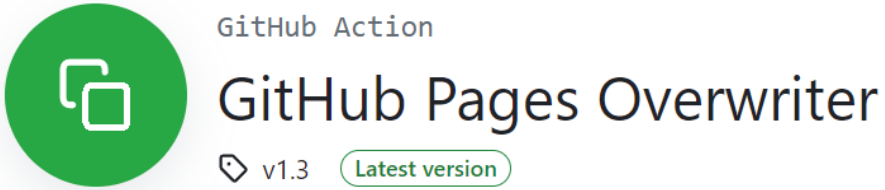
- Generates a documentation website (no deployment, however) automatically when you push to dev or to main
  - The docs generated for the dev branch and the docs generated for the main branch are stored in separate folders.
- Can easily configure the template for the documentation website (i.e. its appearance – see here)
- To use, use /** JSDoc comment */ in your code (above classes, functions, and variables)
  - Recommended: use tags and specify the types of parameters:

```
/**
 * Creates a FortuneEngine object.
 * @param {string} app_name that gives name of fortune teller type
 */
```

# Minimization – UglifyJS + HTML Minifier

- Minimization makes the distributed code smaller by removing whitespace and shortening variable names
- Minifies files in src/ and puts the minified versions in dist/
  - Code (HTML, CSS, JS) gets minified and put in dist/
  - All other files (e.g. JSON, pictures, sound files, etc.) simply get copied
  - All relevant files **MUST** be in src folder
    - That means that we need to move the assets/ folder into the src/ folder!
- Only gets run when pushing to main
- Originally run with a GitHub Action – now, it is accomplished via running a shell script (minimize.sh) stored in the .github/ directory

# Deployment – GitHub Pages




- Deployment hosts our code from the dist/ directory onto GitHub Pages
- This is the last step (after minimization) of the push to main workflow
- Deploys whatever is in the dist/ directory onto a GitHub Page
  - It does this by pushing the contents of the dist/ directory to the gh-pages branch
  - GitHub then runs its own workflow to deploy the contents of the gh-pages branch to https://cse110-sp23-group21.github.io/map-my-future/

# Let’s take a look at the .yml files