University of York Department of Computer Science
Engineering 1

Continuous Integration Report

Cohort 3 Team 5 - alltheeb5t

Aaron Heald
Alex Gu
Arun Hill
Jade Stokes
Maksim Soshchin
Meg Tierney
Will Hall

The continuous integration strategy implemented for this project can be described roughly by two primary objectives: **preserving code integrity** and **preventing significant branch divergence**.

Preserving Code Integrity

- ### Automated unit test runners
  Triggers: Any commit to `main` **Inputs** Code (everything in `core` package), Unit Tests (housed in `headless` package) **Output:** Overall result + full summary of passed / failed tests
  Automated unit tests (JUnit) are only triggered for commits to the `main` branch since we employed a test-driven development strategy, with other branches containing tests for not-yet completed features.

  We didn't want the failure of these tests to provide a misleading indication that the code was incorrect. After making changes, people would run the tests locally on their machine, providing an initial opportunity to correct problems. The automated test runners are designed only as a fallback, alerting people of problematic code and providing a reference machine.
- ### Code Reviews
  Triggers: When people decide that they've made sufficient changes to merge
  Input: Code (with new changes highlighted) **Output:** Comments for changes / accept PR
  When people had a working implementation of a feature (even if only a subsection of a larger feature), they were encouraged to merge this into main in order to catch integration issues as soon as possible. Merges were performed via a pull request and, for any sizable changes, they were reviewed by a second person. This served as a way to identify unclear code, spot possible bugs and also to increase the development team's BUS factor (ie. the reviewer would gain at least a basic understanding of how the code works and be in a better position to take over development if necessary).
- ### Automated Building
  Triggers: Any commit to main **Inputs:** Entire codebase at the current moment in time
  Output: A single .jar file that can be downloaded independently of the main code
  This meant that it was easy for other team members to download and test the latest working version, allowing them to provide feedback and spot newly introduced problems.

Preventing Branch Divergence

- ### Automatic Merge Script
  Triggers: Any commit to main **Inputs:** Entire codebase (from main), List of branches to merge into
  Output: Updated codebase on all branches / PRs created for manual review
  In the lectures, we were introduced to Martin Fowler's best practices for Continuous integration[1], which included everyone pushing to main directly. We decided that having separate branches and a frequent pull request schedule would be more appropriate given the experience and timescale of our project as it allowed people to push their [not yet working] changes to continue elsewhere and meant that any problematic merges could be handled centrally by the most appropriate person.

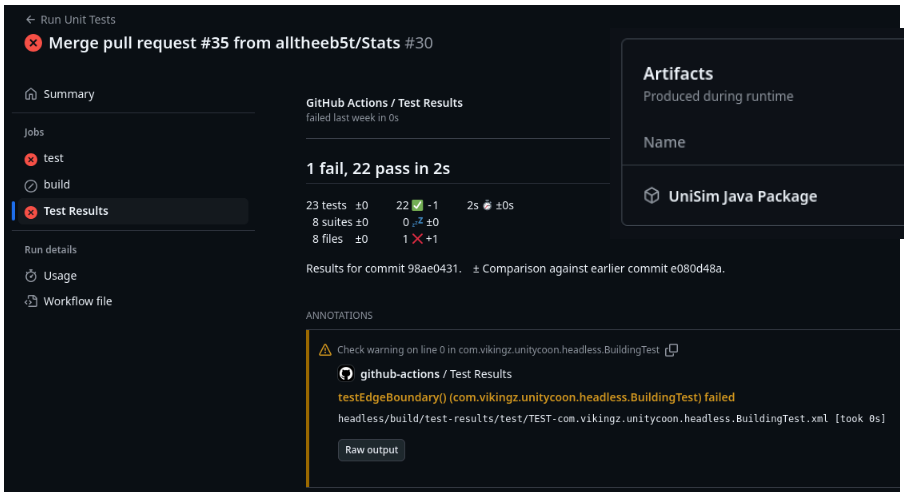
  We have a continuous integration workflow that, on pushes to `main`, would automatically merge those changes into active development branches (or open a pull request if conflicts occurred). This is described by Martin Fowler as Semi-Integration[1] as it ensures that any stable changes made by other people are introduced quickly into active development branches and reduces friction when said branch is merged into main.

Infrastructure Details

For our continuous integration, we made use of GitHub Actions because we were already using GitHub for our code storage, it is well documented and costs nothing. Additionally, members of our team had some prior experience with this tool. When developing the actions, we implemented matters of best practice such as using the SHA as a version identifier on 3rd party actions and using the permissions keyword to restrict an action's ability to the minimum required.

Action 1: Unit testing & automated builds (*unit-tests.yaml on website* / Appendix. 1)

- In this action, a simple sequential architecture is used.
- The latest version of the repository is checked out and Gradle is setup (these steps were documented in the github actions examples).
- The action then executes the tests and uses a 3rd party action by GitHub user EricoMi to display the results in a useful summary format (see figure. 1).
- The output state (success/failure) of the 'Publish Test Results' job matches the result of the testing itself. This means that a successful **job** can be used to indicate all tests passed.
- In case the job was successful, a build job repeats the Gradle setup steps and runs the `gradlew build` command.
- The generated binary is stored within the file system of the Actions runner so in order to access it, it must be uploaded as an artifact. This is accessible via GitHub (figure. 2)


← Run Unit Tests

Merge pull request #35 from alltheeb5t/Stats #30

Summary

Jobs

test

build

Test Results

Run details

Usage

Workflow file

GitHub Actions / Test Results
failed last week in 0s

1 fail, 22 pass in 2s

```
23 tests  ±0      22 ✅ -1      2s ⏱ ±0s
 8 suites ±0       0 💤 ±0
 8 files   ±0       1 ❌ +1
```

Results for commit 98ae0431.   ± Comparison against earlier commit e080d48a.

ANNOTATIONS

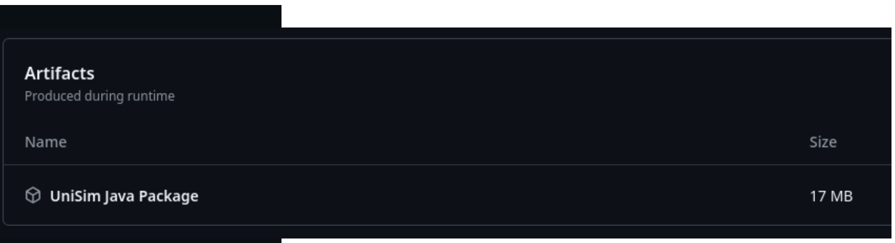
Check warning on line 0 in com.vikingz.unitycoon.headless.BuildingTest

github-actions / Test Results

testEdgeBoundary() (com.vikingz.unitycoon.headless.BuildingTest) failed

```
headless/build/test-results/test/TEST-com.vikingz.unitycoon.headless.BuildingTest.xml [took 0s]
```

Raw output


Artifacts
Produced during runtime

Name	Size
UniSim Java Package	17 MB

Above: figure.2, a screenshot of the download page for the built binary

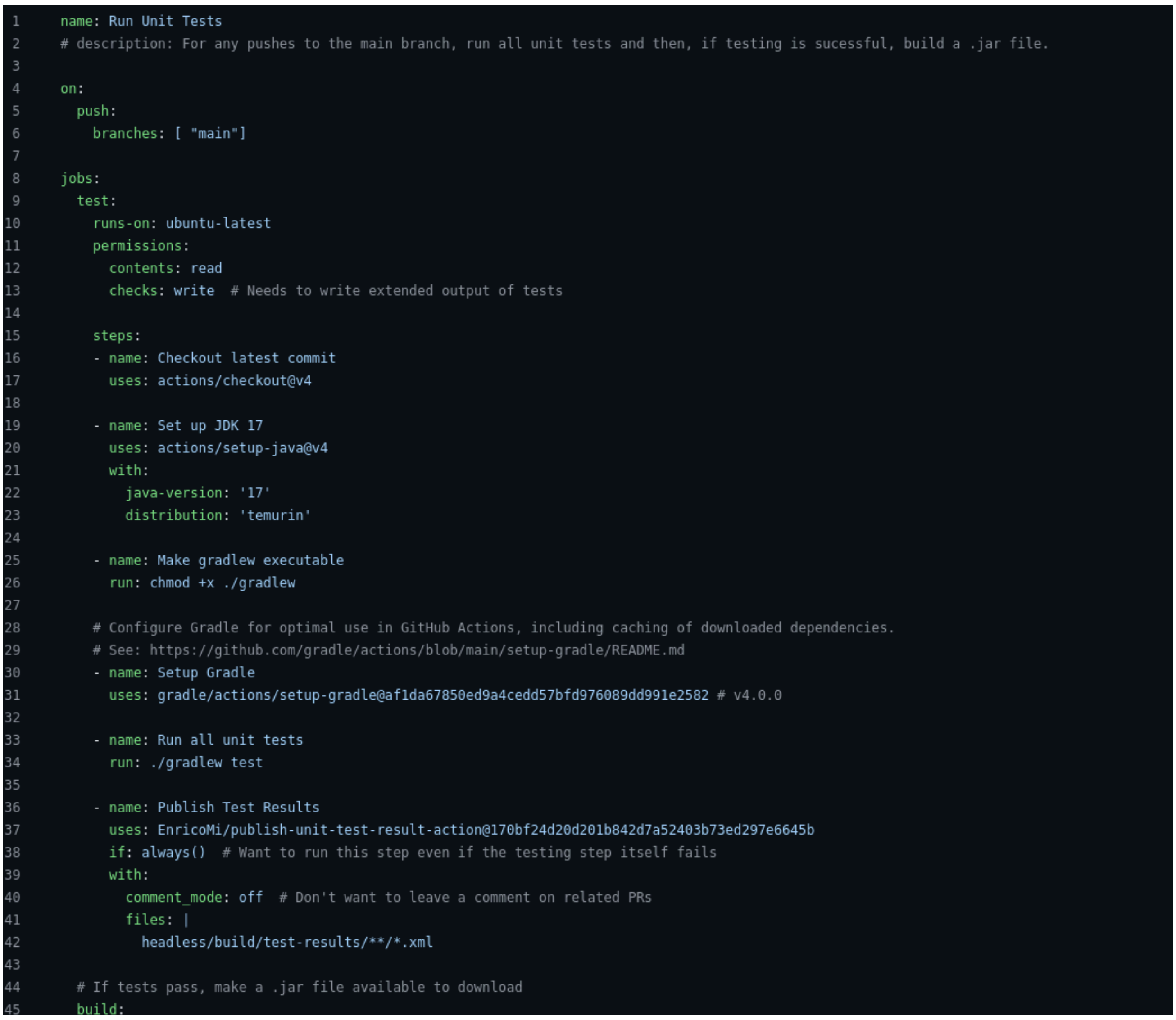
Left: figure.1, a screenshot showing the summary page where one test has failed

Action 2: Automatic Merge (*automatic_merge.yaml* in Appendix. 1)

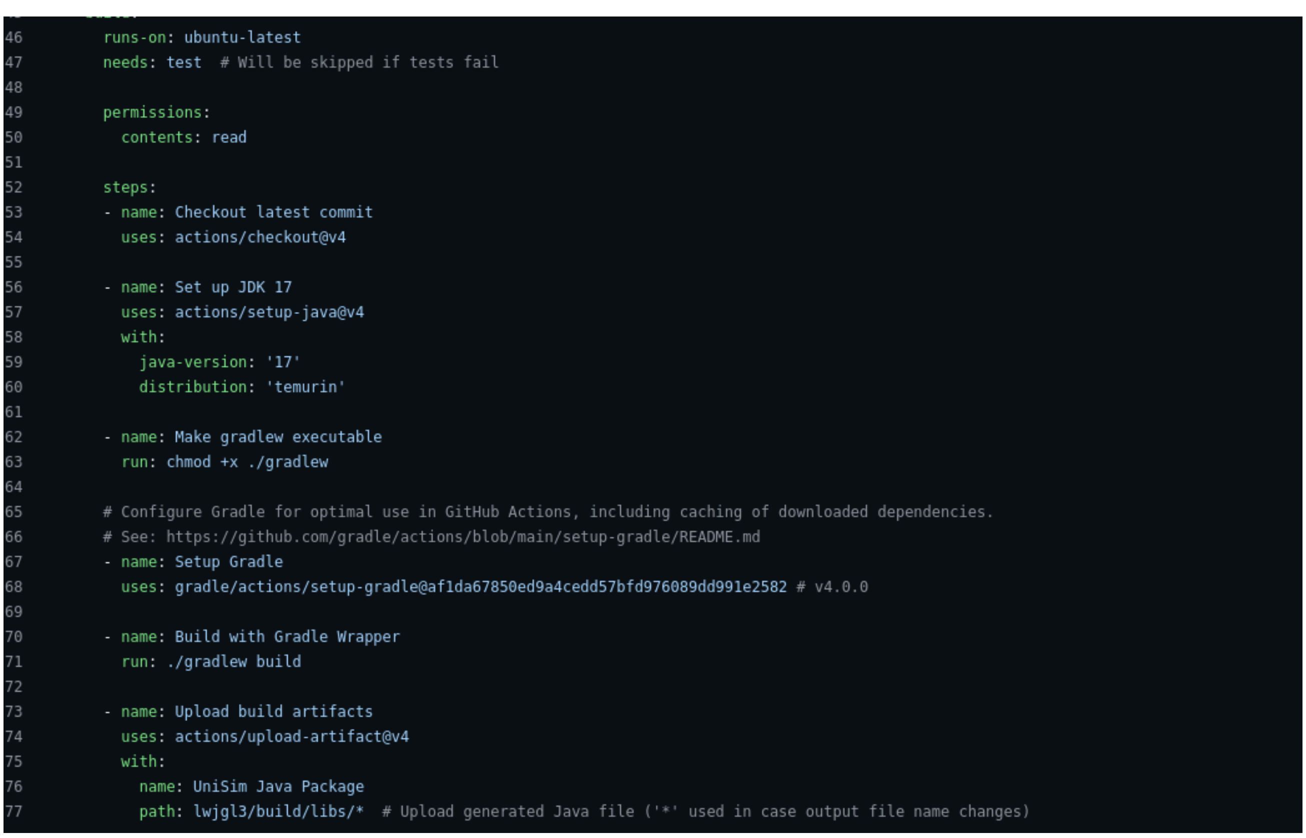
- There are two development branches into which new changes are merged. We were unable to define a list of branches dynamically using a wildcard so had to hardcode names for two branches. Use of the matrix strategy would allow additional branches to be added easily. The following steps are performed in parallel for each of the defined branches
- An attempt is made to merge using a simple `git merge` command. This is designed to be used in case of disjoint changes.
- In case there is **any** conflict, the merge fails. It is best to resort to manual input so as to prevent introduction of errors. To do this, the action uses an action by GitHub user peter-evans to create a new pull request. A review is requested from a designated team member who can review and merge the changes ASAP.
- The content of `main` is placed into a temporary `staging` branch so that, when merged, the content of main isn't inadvertently affected.

Appendix 1

unit-tests.yaml

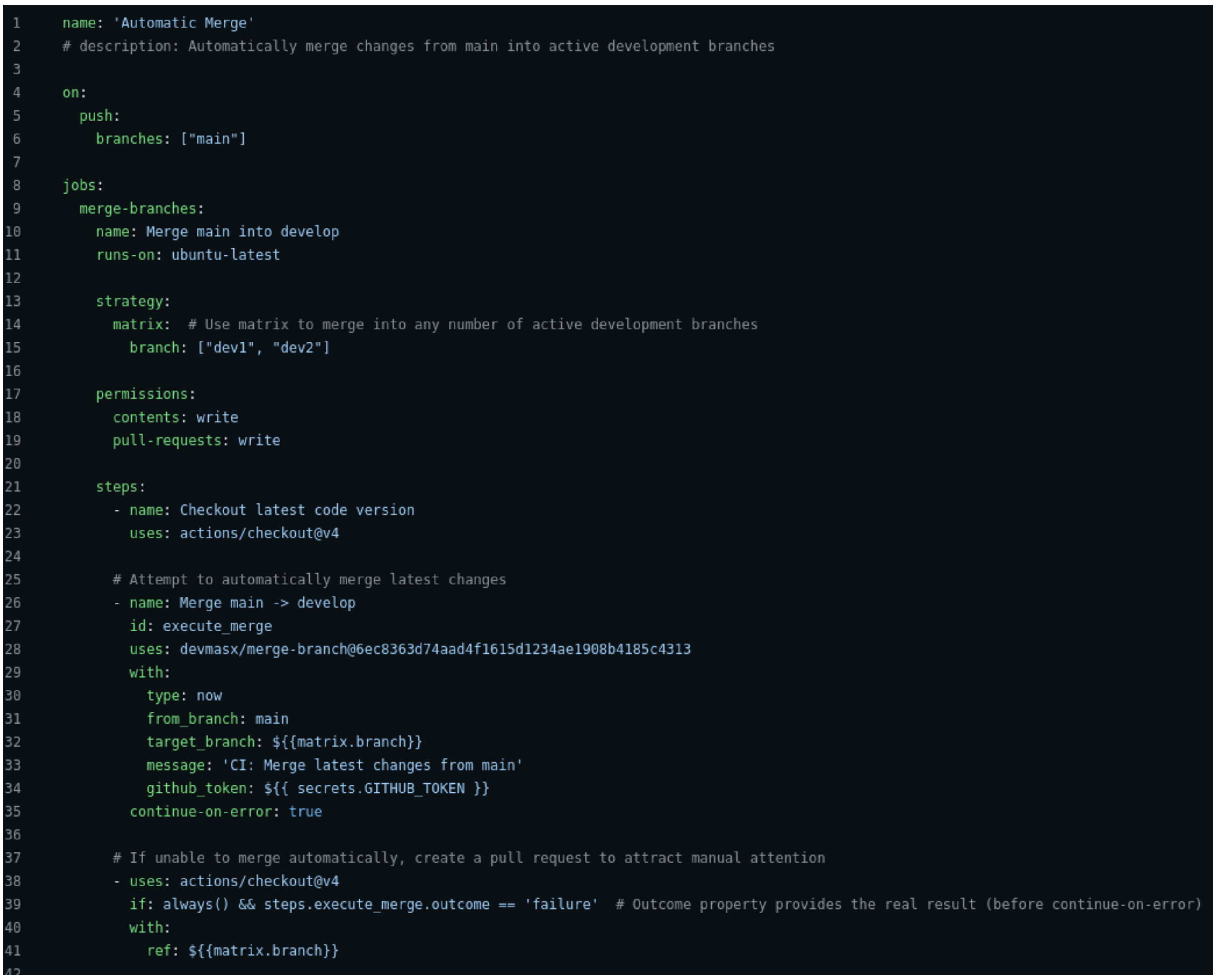


```yaml
name: Run Unit Tests
# description: For any pushes to the main branch, run all unit tests and then, if testing is sucessful, build a .jar file.

on:
  push:
    branches: [ "main"]

jobs:
  test:
    runs-on: ubuntu-latest
    permissions:
      contents: read
      checks: write  # Needs to write extended output of tests

    steps:
    - name: Checkout latest commit
      uses: actions/checkout@v4

    - name: Set up JDK 17
      uses: actions/setup-java@v4
      with:
        java-version: '17'
        distribution: 'temurin'

    - name: Make gradlew executable
      run: chmod +x ./gradlew

    # Configure Gradle for optimal use in GitHub Actions, including caching of downloaded dependencies.
    # See: https://github.com/gradle/actions/blob/main/setup-gradle/README.md
    - name: Setup Gradle
      uses: gradle/actions/setup-gradle@af1da67850ed9a4cedd57bfd976089dd991e2582 # v4.0.0

    - name: Run all unit tests
      run: ./gradlew test

    - name: Publish Test Results
      uses: EnricoMi/publish-unit-test-result-action@170bf24d20d201b842d7a52403b73ed297e6645b
      if: always()  # Want to run this step even if the testing step itself fails
      with:
        comment_mode: off  # Don't want to leave a comment on related PRs
        files: |
          headless/build/test-results/**/*.xml

  # If tests pass, make a .jar file available to download
  build:
```

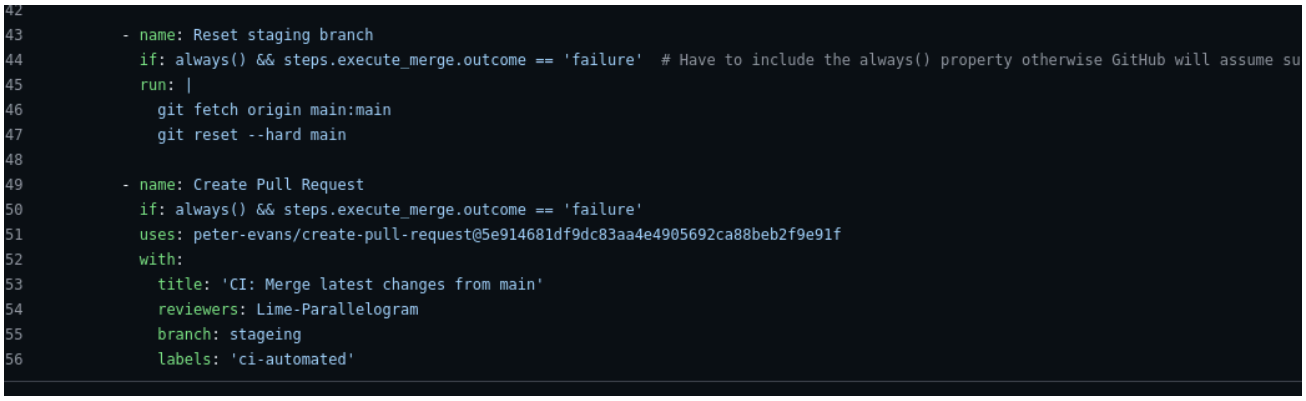


```yaml
    runs-on: ubuntu-latest
    needs: test  # Will be skipped if tests fail

    permissions:
      contents: read

    steps:
    - name: Checkout latest commit
      uses: actions/checkout@v4

    - name: Set up JDK 17
      uses: actions/setup-java@v4
      with:
        java-version: '17'
        distribution: 'temurin'

    - name: Make gradlew executable
      run: chmod +x ./gradlew

    # Configure Gradle for optimal use in GitHub Actions, including caching of downloaded dependencies.
    # See: https://github.com/gradle/actions/blob/main/setup-gradle/README.md
    - name: Setup Gradle
      uses: gradle/actions/setup-gradle@af1da67850ed9a4cedd57bfd976089dd991e2582 # v4.0.0

    - name: Build with Gradle Wrapper
      run: ./gradlew build

    - name: Upload build artifacts
      uses: actions/upload-artifact@v4
      with:
        name: UniSim Java Package
        path: lwjgl3/build/libs/*  # Upload generated Java file ('*' used in case output file name changes)
```

automatic_merge.yaml



```yaml
name: 'Automatic Merge'
# description: Automatically merge changes from main into active development branches

on:
  push:
    branches: ["main"]

jobs:
  merge-branches:
    name: Merge main into develop
    runs-on: ubuntu-latest

    strategy:
      matrix:  # Use matrix to merge into any number of active development branches
        branch: ["dev1", "dev2"]

    permissions:
      contents: write
      pull-requests: write

    steps:
      - name: Checkout latest code version
        uses: actions/checkout@v4

      # Attempt to automatically merge latest changes
      - name: Merge main -> develop
        id: execute_merge
        uses: devmasx/merge-branch@6ec8363d74aad4f1615d1234ae1908b4185c4313
        with:
          type: now
          from_branch: main
          target_branch: ${{matrix.branch}}
          message: 'CI: Merge latest changes from main'
          github_token: ${{ secrets.GITHUB_TOKEN }}
        continue-on-error: true

      # If unable to merge automatically, create a pull request to attract manual attention
      - uses: actions/checkout@v4
        if: always() && steps.execute_merge.outcome == 'failure'  # Outcome property provides the real result (before continue-on-error)
        with:
          ref: ${{matrix.branch}}
```



```yaml
      - name: Reset staging branch
        if: always() && steps.execute_merge.outcome == 'failure'  # Have to include the always() property otherwise GitHub will assume su
        run: |
          git fetch origin main:main
          git reset --hard main

      - name: Create Pull Request
        if: always() && steps.execute_merge.outcome == 'failure'
        uses: peter-evans/create-pull-request@5e914681df9dc83aa4e4905692ca88beb2f9e91f
        with:
          title: 'CI: Merge latest changes from main'
          reviewers: Lime-Parallelogram
          branch: stageing
          labels: 'ci-automated'
```

References

[1] M. Fowler. (2024, Dec. 21). *Continuous Integration* [Online]. Available: https://martinfowler.com/articles/continuousIntegration.html