Controlling Execution Flow in GitHub Actions

GitHub Actions provides flexible ways to control the execution flow of workflows, allowing you to define conditions for when to run specific steps or jobs and specify dependencies between them. This README explains the key concepts and techniques for controlling execution flow in GitHub Actions.

Conditional Execution of Steps and Jobs

Using the if Expression

GitHub Actions allows you to conditionally execute steps or entire jobs using the `if` expression. The `if` expression evaluates a condition and runs the associated step or job if the condition is true.

Example:

```
jobs:
  build:
    runs-on: ubuntu-latest

    steps:
    - name: Build and Test
      if: github.event_name == 'push'
      run: |
        # Your build and test commands here
        echo "Building and testing..."
```

In this example, the step "Build and Test" only runs when the workflow is triggered by a push event.

Contextual Expressions

You can use contextual expressions within if conditions to access information about the workflow run, event, or repository. These expressions include github.event_name, github.ref, and more.

```
jobs:
  deploy:
    runs-on: ubuntu-latest
    steps:
    - name: Deploy to Production
      if: github.event_name == 'push' && github.ref == 'refs/heads/main'
      run: |
        # Deployment steps for main branch
        echo "Deploying to production..."
```

In this example, the "Deploy to Production" step only runs when a push event occurs on the main branch.

Defining Dependencies Between Jobs

GitHub Actions enables you to specify dependencies between jobs using the `needs` keyword. When a job depends on another job, it waits for the dependent job to complete before starting.

```
jobs:
  build:
    runs-on: ubuntu-latest
    steps:
    - name: Build and Test
      run: |
        # Your build and test commands here
        echo "Building and testing..."

  deploy:
    needs: build
    runs-on: ubuntu-latest
    steps:
    - name: Deploy to Staging
      run: |
        # Deployment steps for staging environment
        echo "Deploying to staging..."
```

In this example, the "deploy" job depends on the "build" job, ensuring that the code is built and tested before deployment.

Testing Execution Flow in Workflow

1. Create Workflow File:

- Create a new file named `12-01-Execution-Flow.yml` in the `.github/workflows` directory of your repository.

  ```
  name: 12-01-Controlling the Execution Flow

  on:
    workflow_dispatch:
      inputs:
        pass-unit-tests:
          type: boolean
          description: Whether unit tests will pass or not
          default: true
  ```

```yaml
jobs:
  lint-build:
    runs-on: ubuntu-latest
    steps:
      - name: Lint and build
        run: echo "Linting and building project"
  unit-tests:
    runs-on: ubuntu-latest
    steps:
      - name: Running unit tests
        run: echo "Running tests..."
      - name: Failing tests
        if: ${{ !inputs.pass-unit-tests }}
        run: exit 1
  deploy-nonprod:
    runs-on: ubuntu-latest
    needs:
      - lint-build
      - unit-tests
    steps:
      - name: Deploying to nonprod
        run: echo "Deploying to nonprod..."
  e2e-tests:
    runs-on: ubuntu-latest
    needs:
      - deploy-nonprod
    steps:
      - name: Running E2E tests
        run: echo "Running E2E tests"
  load-tests:
    runs-on: ubuntu-latest
    needs:
      - deploy-nonprod
    steps:
      - name: Running load tests
        run: echo "Running load tests"
  deploy-prod:
    runs-on: ubuntu-latest
    needs:
      - e2e-tests
      - load-tests
    steps:
      - name: Deploying to prod
        run: echo "Deploying to prod..."
```

- Copy and paste the provided YAML configuration into this file.

2. Understanding the Workflow:

- This workflow is triggered manually via the GitHub UI (`workflow_dispatch`).
- It includes multiple jobs (`lint-build`, `unit-tests`, `deploy-nonprod`, `e2e-tests`, `load-tests`, `deploy-prod`) that represent different stages of the deployment pipeline.

- Each job runs on an Ubuntu latest runner and performs specific tasks related to linting, testing, deployment, and end-to-end (E2E) and load testing.
- Conditional execution is controlled using the `needs` keyword, ensuring that jobs are executed in the correct order based on their dependencies.
- The `unit-tests` job includes a condition based on the value of the pass-unit-tests input, allowing for the simulation of passing or failing unit tests.

3. Testing the Workflow:

- Commit and push the workflow file (`12-01-Execution-Flow.yml`) to your repository.
- Navigate to the "Actions" tab in your GitHub repository.
- Manually trigger the workflow by clicking on the "Run workflow" button for the `12-01-Execution-Flow` workflow.
- When prompted, choose whether to pass or fail unit tests by providing the `pass-unit-tests` input value.
- Monitor the workflow run and review the execution of each job and step.
- Ensure that jobs are executed in the correct order and that conditional steps are triggered based on the input value provided.

4. Observing Output:

- Observe the logs of each job and step to verify that tasks are performed as expected.
- Pay attention to the conditional step in the `unit-tests` job and confirm that it behaves correctly based on the value of the `pass-unit-tests` input.

Best Practices

- **Keep workflows modular:** Break down workflows into smaller jobs and steps, making it easier to manage conditions and dependencies.
- **Use descriptive step and job names:** Clear names make it easier to understand the purpose of each part of your workflow.
- **Test workflows thoroughly:** Ensure that conditional logic and job dependencies work as expected by testing different scenarios.