Announcements

- Team Formation Survey Due 5/16
- AI in SWD extra credit survey/assignment due 5/18
- Quiz 1 in class today for accuracy – based on the lecture
- Sign up for Google Cloud Credits. Details in an Ed post.
- You should be added to my organization on Open AI. Accept the invite soon. Invite expires every 3 days.
- Sign up for GitHub Pro for students

CS3300 A: Introduction to Software Engineering

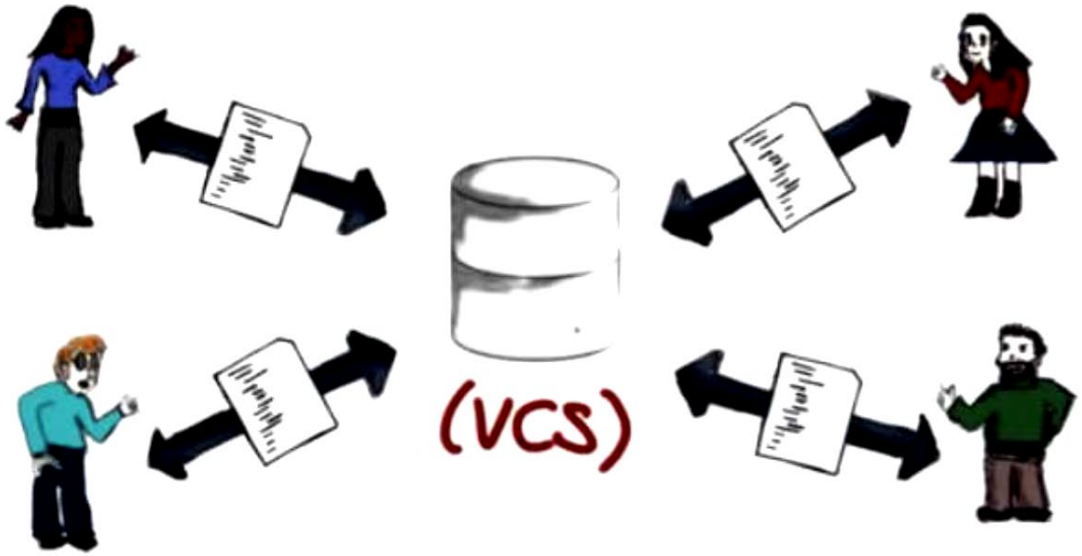
Lecture 02: Tools of the Trade #1

Version Control Systems, GIT, Code Review, GitHub Actions

Dr. Nimisha Roy ▸ nroy9@gatech.edu

Git Refresher

What are Version Control Systems?



- A tool that software developers use for keeping track of revisions of their project
  - snapshots of your project over time.
  - Documents, source files etc.
- Most obvious benefits:
  - Option to go back and revisit
  - Collaborate with multiple people

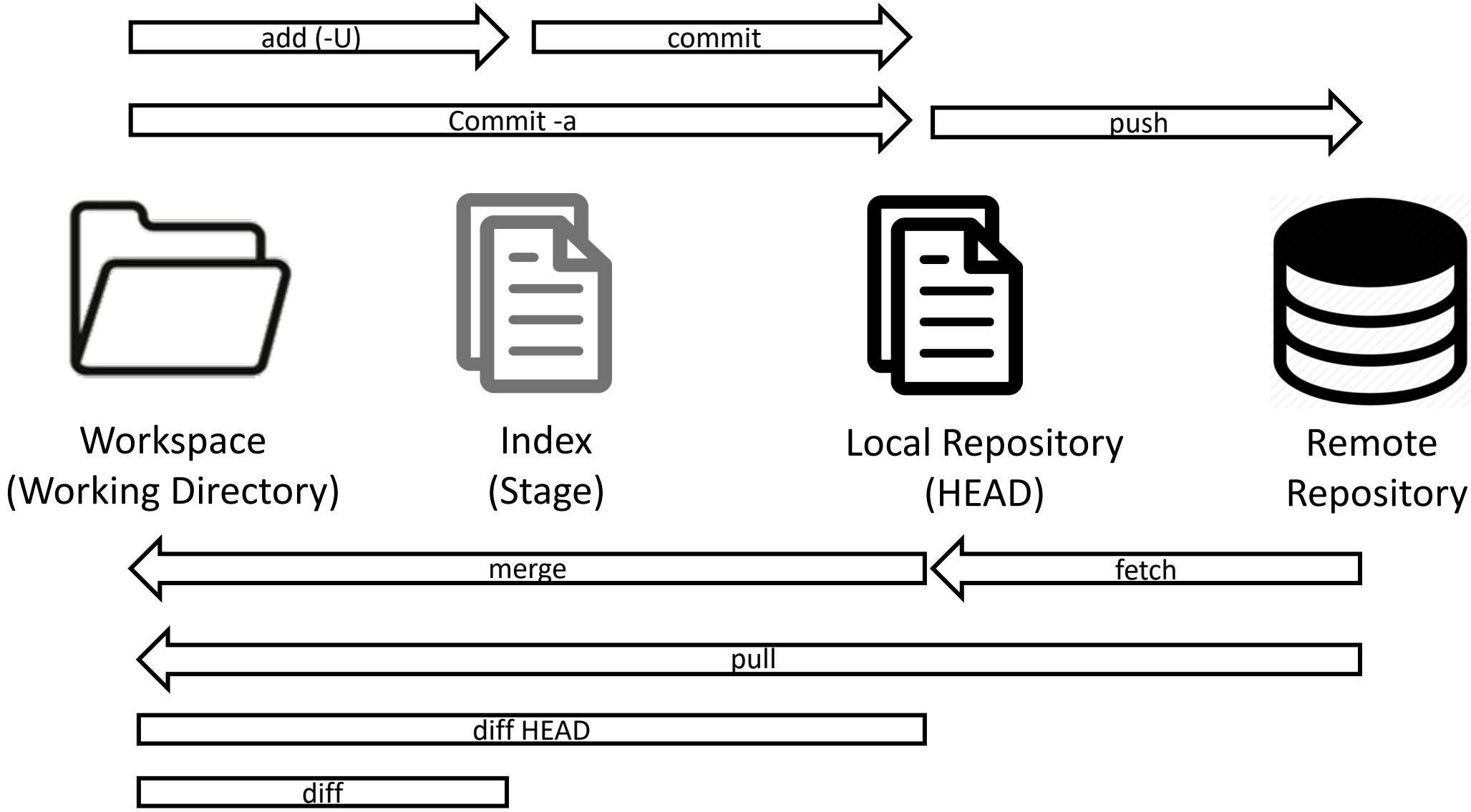
Importance

- **Enforce Discipline**: Manages process by which control of items passes from one person to another
- **Archive versions**: store subsequent versions of source-controlled items
- **Maintain Historical Information**: Author of a specific version; date and time of a specific version; etc. Retrieve and compare.
- **Enables Collaboration**: share data, files and documents
- **Recover from accidental edits/revisions**
- **Conserve Disk Space**: centralizing the management of the version.
  - Instead of having many copies spread around, one or a few central points where these copies are stored
  - efficient algorithms to store changes, so keep many versions without taking up too much space.

Don’ts in VCS

- Adding Derived Files
  - E.g., executable file derived from compiling a set of source codes
  - No reason to add it
- Adding bulky binary files
  - Try to keep them local
- Creating a local copy of files/tree of files
  - Don’t do this!!
  - Useless, risky, confusing
  - Trust the version control system

GIT Workflow Recap

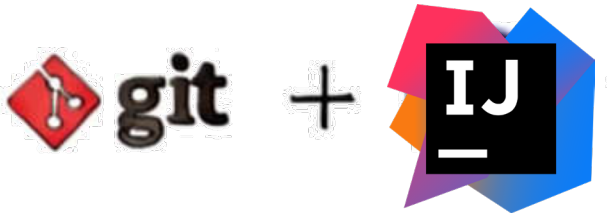


GIT Plugins

Install GIT: Follow instructions on https://git-scm.com/book/en/v2/Getting-Started-Installing-Git

Egit: GIT Plugin available for Eclipse; can be downloaded at www.eclipse.github.com and can be installed in Eclipse

GitToolBox for IntelliJ: The plugin can be downloaded here.



Visual Studio Code has integrated source control management (SCM) and includes Git support out-of-the-box.

GitHub

- GIT hosting website. Get an account and create your remote repositories
- GitHub repository for your projects
- Provides easy-to-use FREE desktop clients for Mac and Windows (https://desktop.github.com)

- **GitHub Pages**:
  - One click to enable for your GitHub repo.
  - Hosted directly from your GitHub repository.
  - Just edit, push, and your changes are live.
  - This course's website is a GitHub Page.

- ALWAYS SET YOUR GITHUB REPOSITORY TO BE PRIVATE, UNLESS YOU ARE ABSOLUTELY SURE YOU WANT IT PUBLIC !!!

Git Basics Demo notes on website

Branches, Merge Conflict, Code Review

GIT Demo – Creating Branches

- By default, when you create your project you will be on main/master
- It is good practice to have different branches for different features, people, etc.
- To see all local branches:
  - git branch
- To create a new branch:
  - git branch [BRANCHNAME]
- To move to (checkout) a branch:
  - git checkout [BRANCHNAME]
- To create a new branch and move to it:
  - git checkout –b [BRANCHNAME]

GIT Demo – Merging Branches

- Merging allows you to carry the changes in one branch over to another branch, combining both branches

To merge two branches:

1. git checkout [NAME_OF_BRANCH_TO_MERGE_INTO]
2. git merge [NAME_OF_BRANCH_TO_BRING_IN]

Example: merging *feature* branch into *master* branch:

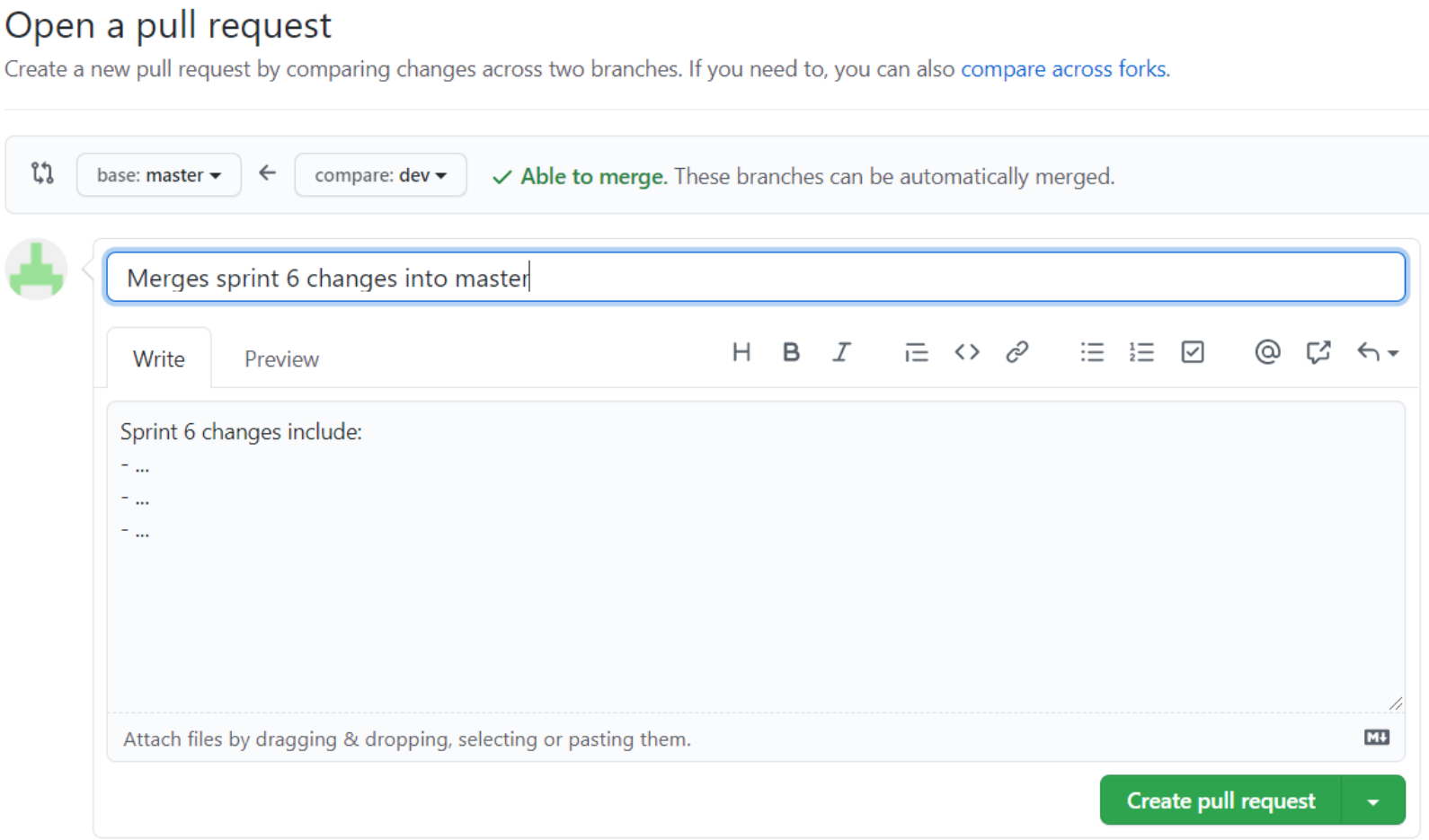
1. git checkout master
2. git merge feature

Why Code Reviews?

- Improve Overall Quality of Code
  - Having eyes on source code that you didn't write can help identify issues
- Facilitating Team Collaboration
  - Checking out each other's code better helps you understand how each feature is implemented
- Identifying bugs early in process
- Good for onboarding new developers to establish best practices within the organization
- Significant % of your time in your job is code maintenance.

Pull Requests

- Tool to aggregate branch changes and request that the changes be merged into a different branch.
- Done through the GitHub GUI



Branch Protection

Protected branches ensure that collaborators on your repository cannot make irrevocable changes to branches. Enabling protected branches also allows you to enable other optional checks and requirements, like required status checks and required reviews.

Always have a branch protection rule enforced in your main GitHub repository branch. Settings → Branches → Branch Protection Rule → Require a pull request before merging

Note: This is only possible for public repositories (with GitHub free) and private repositories (with GitHub Pro). **So, sign up for GitHub Pro (https://education.github.com/discount_requests/application)**

Code Review Assignment: Creating Branches, Pull Requests, Performing Reviews

Both for Projects 1 and 2:

1. Create Separate Branches for every feature
   a. You might create sub branches of these branches as you implement new portions of each feature
2. Perform a Pull Request
3. Reviewing Code and Closing Pull Requests
4. Merging Branches [Do not delete them until the assignment is graded]

Let's do a quick demo of these items.

5 min break

GitHub Actions

GitHub Actions

- GitHub Actions is an event-driven workflow automation product for supporting the software development process in the GitHub environment.

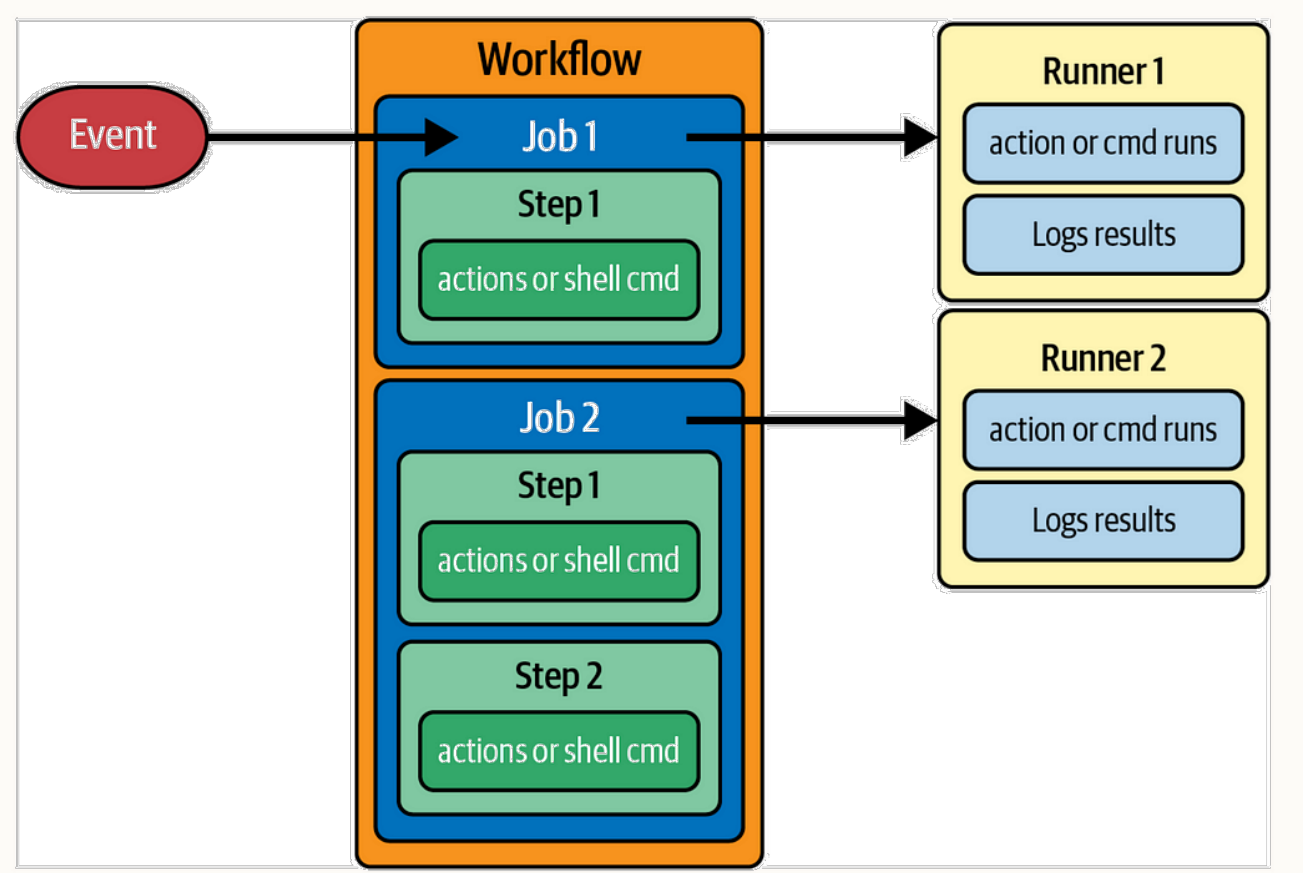
- Using GitHub Action , we can:
  - Automate SDLC (Software Development Lifecycle) workflows
  - Implement CI/CD, DevOps

Anatomy of a GitHub Action

```
name: Example workflow

on: push

jobs:

  build:
    runs-on:

    steps:

      - uses: actions/checkout@v2

      ...
```

Anatomy of a GitHub Action

- A workflow is a unit of automation from start to finish. It contains one or more jobs.
- A workflow is triggered by an event.
- Jobs, in turn are made up of steps.
- A step either runs a shell command or invokes a predefined github **action**. All of the steps in a job are executed on a *runner*.
- The runner is a server(virtual or physical) or a container that has been setup to understand how to interact with GitHub Actions.



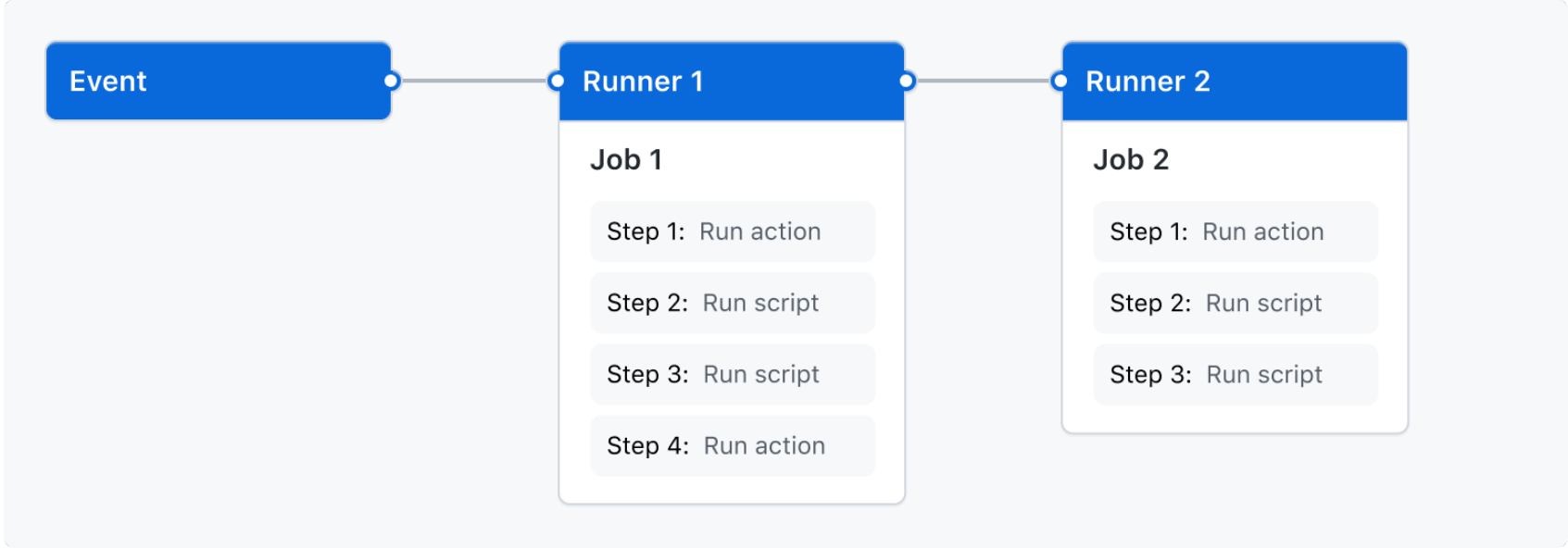
Anatomy of a GitHub Action

A **workflow** is a unit of automation from its start to finish, including the definition of what triggers the automation (**event**), what environment or other aspects should be taken into account during the automation, and what should happen due to the trigger. Workflow written in YAML format. Examples of events include forking a repository, pushing code to a remote branch, or opening a pull request.

A **job** is a section of the workflow and is made up of one or more steps that execute on the same runner/server (`ubuntu-latest` is the fastest, and cheapest, job runner available.)

A **step** represents one effect of the automation. Each step consists of either a shell script that's executed, or a reference to an action that's run. When we talk about an **action** (with a lowercase "a") in this context, we mean a reusable unit of code provided to GitHub, Actions published by the community, or custom actions defined for specific workflows.

Anatomy of a GitHub Action



- Your workflow contains one or more jobs which can run in sequential order or in parallel.
- Each job will run inside its own runner and has one or more steps that either run a script that you define or run an action, which is a reusable extension that can simplify your workflow.
- Steps are executed in order and are dependent on each other. Since each step is executed on the same runner, you can share data from one step to another. For example, you can have a step that builds your application followed by a step that tests the application that was built.
- An **action** performs a complex but frequently repeated task. An action can pull your Git repository from GitHub, set up the correct toolchain for your build environment, or set up the authentication to your cloud provider.

Events (using on:)

- Single event: `on: push`
- The workflow can respond to a list (multiple events): `on: [push, pull_request]`
- The workflow can respond to event types with qualifiers, such as branches, tags, or file paths:

```
on:
push:
      branches:
      - main
      - 'rel/v*'
      tags:
      - v1.*
      - beta
      paths:
      - '**.ts'
```

- The workflow can execute on a specific schedule or interval ():
- `on:using standard cron syntax`
  `scheduled:`
  `- cron: '30 5,15 * * *'`
- The workflow can respond to specific manual events: `on: [workflow-dispatch, repository-dispatch]`
- The workflow can be called from other workflows: `on: workflow_call`
- The workflow can respond to common activities on GitHub items, such as adding a comment to a GitHub issue: `on: issue_comment`

Steps

- Three basic steps in this workflow.
- These steps
  - check out a set of code,
  - set up a go environment based on a particular version, and
  - run the go process on a source file.
- In the YAML syntax, the ***- character*** indicates where a step starts.
  - The ***uses*** clause indicates that this step invokes a predefined action.
  - The ***with*** clause is used to specify arguments/parameters to pass to the action.
  - And the ***run*** clause indicates a command to be run in the shell.

```
steps:
- uses: actions/checkout@v3
- name: setup Go version
  uses: actions/setup-go@v2
  with:
    go-version: '1.14.0'
- run: go run helloworld.go
```

- Runners are the physical or virtual computers or containers where the code for a workflow is executed. They can be systems provided and hosted by GitHub or they can be instances you set up, host, and control. In a workflow file, runners are defined for jobs simply via the ***runs-on*** clause.

```
runs-on: ubuntu-latest
```

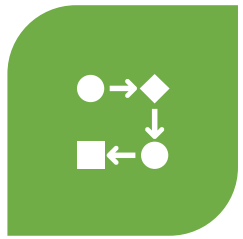
Demo

CREATE A NEW REPOSITORY

SETUP A WORKFLOW (.GITHUB/WORKFLOWS)

ADD A .YML FILE TO THE WORKFLOW. ADD CONTENT

TRIGGER THE EVENT



CHECK THE PROGRESS OF THE WORKFLOW AND JOB USING LOGS

YAML file content

Contents from YAML file copied from:
https://gist.github.com/weibeld/f136048d0a82aacc063f42e684e3c494

01-hello-world.yml

```yaml
name: hello-world
on: push
jobs:
  my-job:
    runs-on: ubuntu-latest
    steps:
      - name: my-step
        run: echo "Hello World!"
```

- `name:` gives your workflow a name. This name will appear in the Actions tab of your repository.
- `on: push:` indicates that your workflow will execute whenever someone pushes to the repository. This is the event
- `my-job:` 1 job triggers on pushing
- `steps:` runs “echo Hello World” on ubuntu terminal that prints it.

YAML file : another example

```yaml
name: Post welcome comment

on:

  pull_request:

          types: [opened]

permissions:

  pull-requests: write

jobs:

  build:

          name: Post welcome comment

          runs-on: ubuntu-latest

          steps:

          - run: gh pr comment $PR_URL --body "Welcome to the repository!"

                    env:

                              GITHUB_TOKEN: ${{ secrets.GITHUB_TOKEN }}

                    PR_URL: ${{ github.event.pull_request.html_url }}
```

- permissions assigns the workflow permissions to operate on the repository
- pull-requests: write gives the workflow permission to write to pull requests. This is needed to create the welcome comment.
- run: gh pr comment $PR_URL --body "Welcome to the repository!“
  - This command uses the gh CLI to post a comment on a pull request.
  - gh pr comment is the command used to add a comment to a pull request.
  - $PR_URL is a variable that holds the URL of the pull request. This URL is used to specify on which pull request the comment should be posted.
  - --body "Welcome to the repository!" specifies the content of the comment.

YAML file: another example

```
build:
      name: Post welcome comment
      runs-on: ubuntu-latest
      steps:
            - run: gh pr comment $PR_URL
--body "Welcome to the repository!"
            env:
                  GITHUB_TOKEN: ${{
secrets.GITHUB_TOKEN }}
                PR_URL: ${{
    github.event.pull_request.html_url
    }}
```

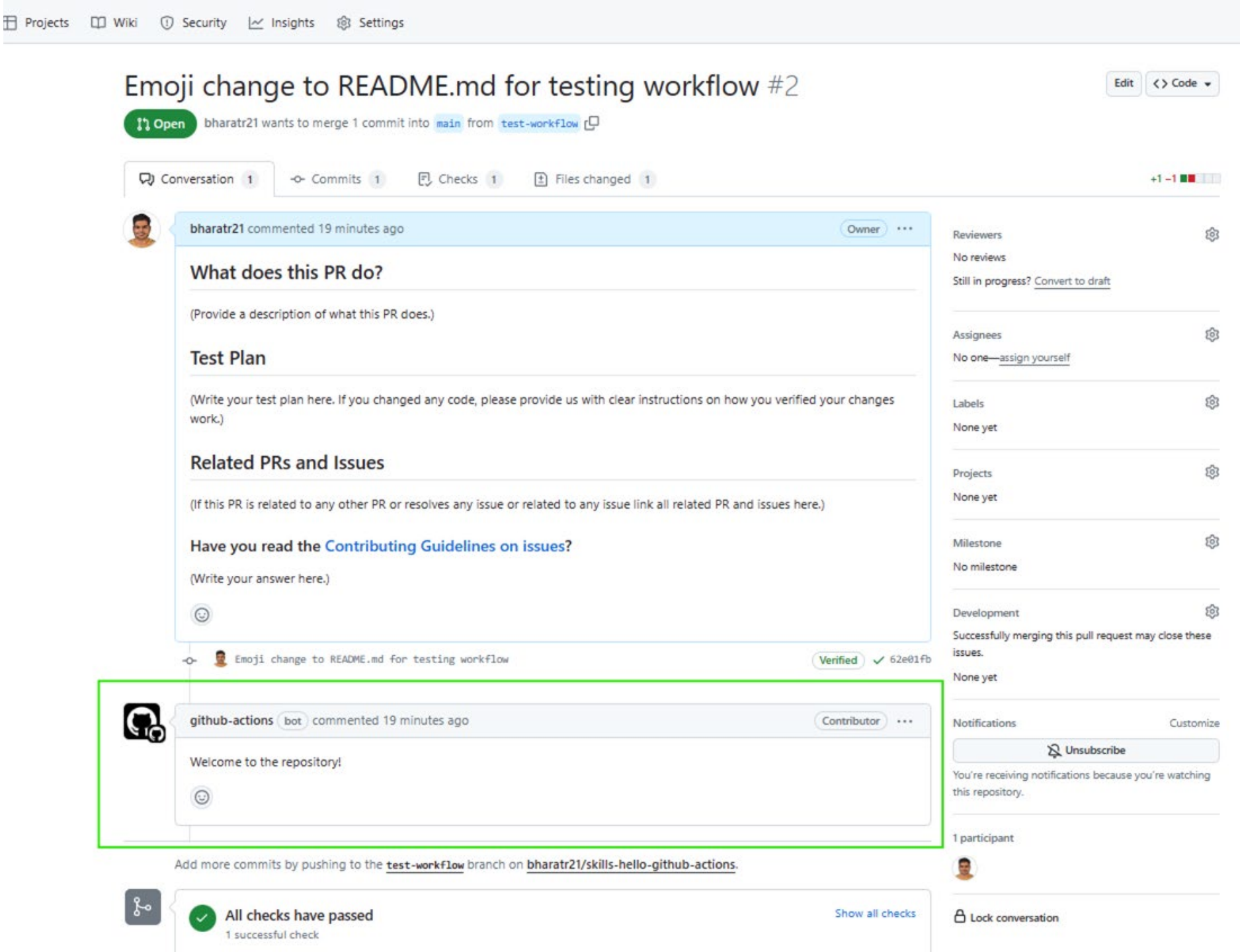
Environment Variables-
GITHUB_TOKEN:GITHUB_TOKEN is used to authenticate with GitHub to carry out actions that require GitHub permissions, such as commenting on a pull request.

- ${{ secrets.GITHUB_TOKEN }} retrieves the token from the repository's encrypted secrets. This token provides the necessary permissions to the GitHub Actions runner to interact with the repository on behalf of the user.
- PR_URL:PR_URL is set to the HTML URL of the pull request that triggered the workflow.
- ${{ github.event.pull_request.html_url }} extracts the URL directly from the event data that triggered the workflow. This ensures that the comment is posted to the correct pull request.

Setting secrets

- Secrets allow you to store sensitive information in your organization, repository, or repository environments.
- Create secrets under Settings → Security → Secrets → Click New repository secret → In the Name field, type a name for your secret → In the Secret field, enter the value for your secret → Click Add secret.
- Example: Add an API Key as a Secret

The Result!


Projects
Wiki
Security
Insights
Settings
Emoji change to README.md for testing workflow #2
Edit
Code
Open
bharatr21 wants to merge 1 commit into main from test-workflow
Conversation 1
Commits 1
Checks 1
Files changed 1
+1 −1
bharatr21 commented 19 minutes ago
Owner
What does this PR do?
(Provide a description of what this PR does.)
Test Plan
(Write your test plan here. If you changed any code, please provide us with clear instructions on how you verified your changes work.)
Related PRs and Issues
(If this PR is related to any other PR or resolves any issue or related to any issue link all related PR and issues here.)
Have you read the Contributing Guidelines on issues?
(Write your answer here.)
Emoji change to README.md for testing workflow
Verified
62e01fb
github-actions bot commented 19 minutes ago
Contributor
Welcome to the repository!
Add more commits by pushing to the test-workflow branch on bharatr21/skills-hello-github-actions.
All checks have passed
1 successful check
Show all checks
Reviewers
No reviews
Still in progress? Convert to draft
Assignees
No one—assign yourself
Labels
None yet
Projects
None yet
Milestone
No milestone
Development
Successfully merging this pull request may close these issues.
None yet
Notifications
Customize
Unsubscribe
You're receiving notifications because you're watching this repository.
1 participant
Lock conversation

GitHub Action for Continuous Integration (CI)

A workflow for linting Markdown files, generating a report in JSON format, and then uploading this report as an artifact.

```
build:
  runs-on: ubuntu-latest
  steps:
     - uses: actions/checkout@v4
     - name: Run markdown lint
       run: |
         npm install remark-cli remark-preset-lint-consistent
         npx remark . --use remark-preset-lint-consistent --frail
```

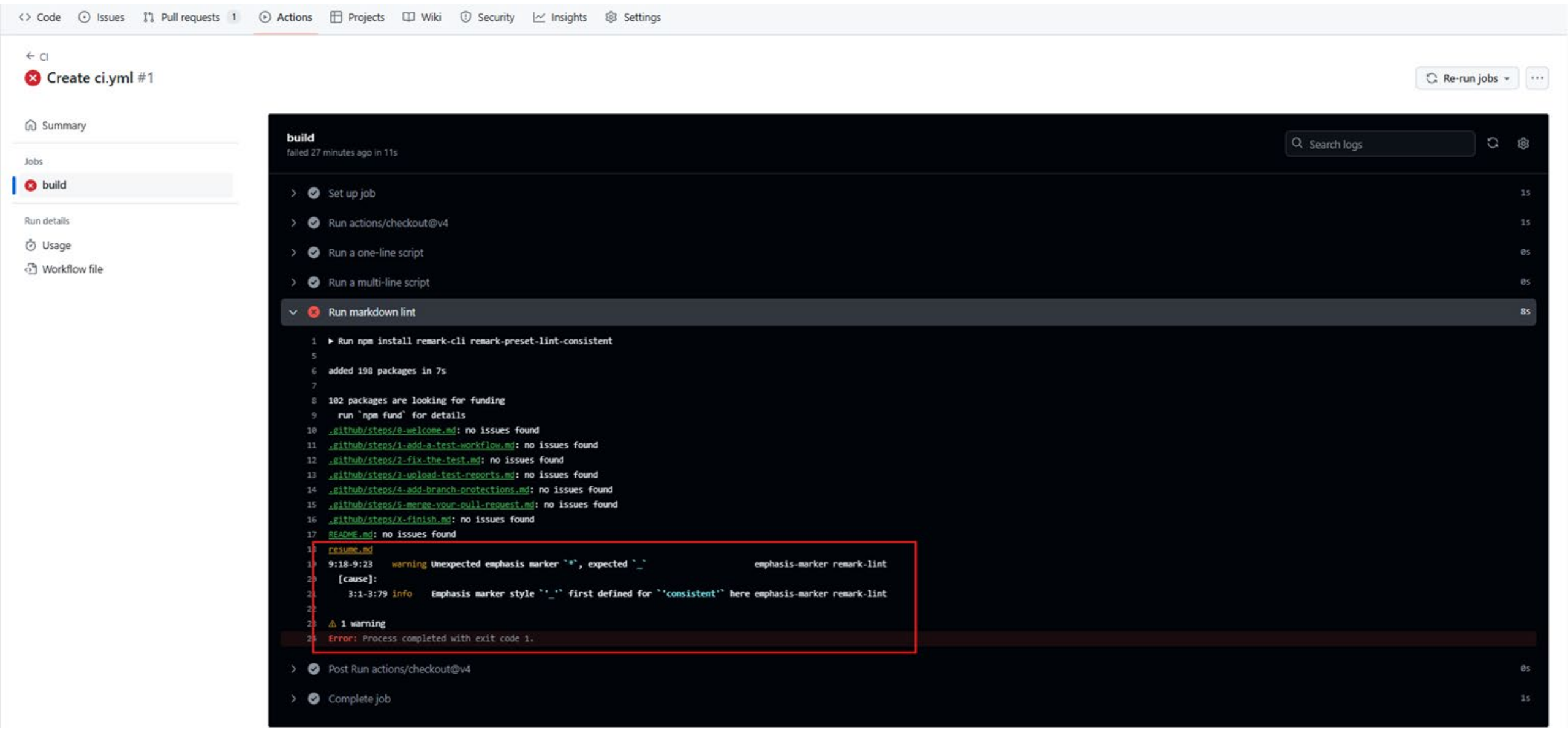
Generating and uploading test reports

When the work product of one job is needed in another (sequential jobs), we can use the built-in artifact storage to save artifacts created from one job to be used in another job within the same workflow.

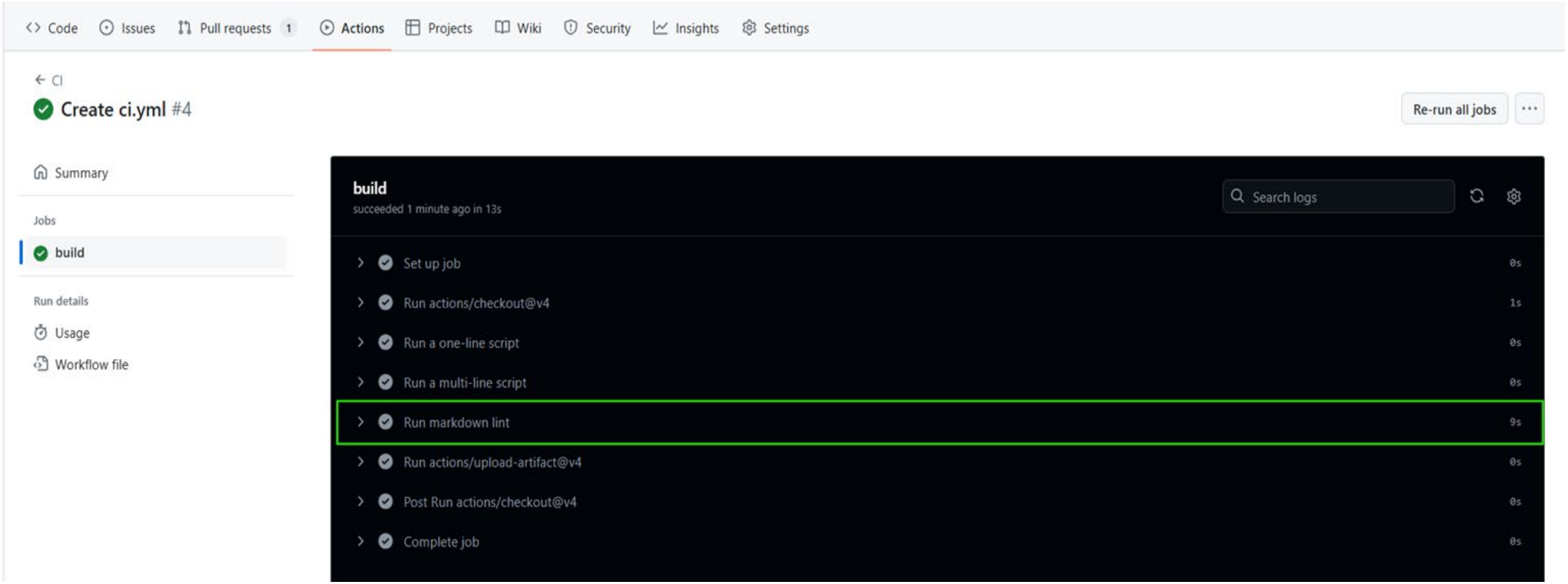
GitHub Action for Continuous Integration (CI)

```
build:
  runs-on: ubuntu-latest
  steps:
    - uses: actions/checkout@v4
    - name: Run markdown lint
      run: |
        npm install remark-cli remark-preset-lint-consistent vfile-reporter-json
        npx remark . --use remark-preset-lint-consistent --report vfile-reporter-json 2> remark-lint-report.json
    - uses: actions/upload-artifact@v4
      with:
        name: remark-lint-report
        path: remark-lint-report.json
```

Failing GitHub Action with logs



Passing GitHub Action



This workflow is ideal for projects that require consistent style and formatting in their Markdown documentation. It ensures that all Markdown files adhere to the specified linting rules, and the lint results are made available for review through the uploaded artifact, which can improve code review processes and maintain code quality.

Summary

- **GitHub Actions**
  - Automates software development workflows directly within GitHub. Triggered by events like push, pull requests, and scheduled times.
  - Can be configured to run on various types of events with precise conditions (e.g., branches, tags).
- **Components of a GitHub Actions Workflow**
  - **Workflows**: Define automated processes from start to finish in a YAML format.
  - **Jobs**: Collections of steps within a workflow.
  - **Steps**: Individual tasks within a job, executed sequentially.
  - **Actions**: Reusable units of code that perform specific functions in a step.
- **Secrets in GitHub Actions**
  - Used to store sensitive information securely. Configurable at repository or organization levels.
  - Critical for maintaining security, especially with API keys and access tokens.
- **Practical Implementation: Benefits of GitHub Actions**
  - **Efficiency**: Automates repetitive and complex tasks, reducing manual effort and increasing productivity.
  - **Reliability**: Ensures consistent execution of deployment and testing workflows, minimizing human errors.
  - **Scalability**: Easily integrates with existing tools and services, supporting both small projects and large-scale operations.
  - **Customization**: Highly customizable to meet specific project needs, from simple notifications to full CI/CD pipelines.

5 min break

Git/GitHub Quiz