Using the GitHub CLI on a runner

How to use advanced GitHub Actions features for continuous integration (CI).

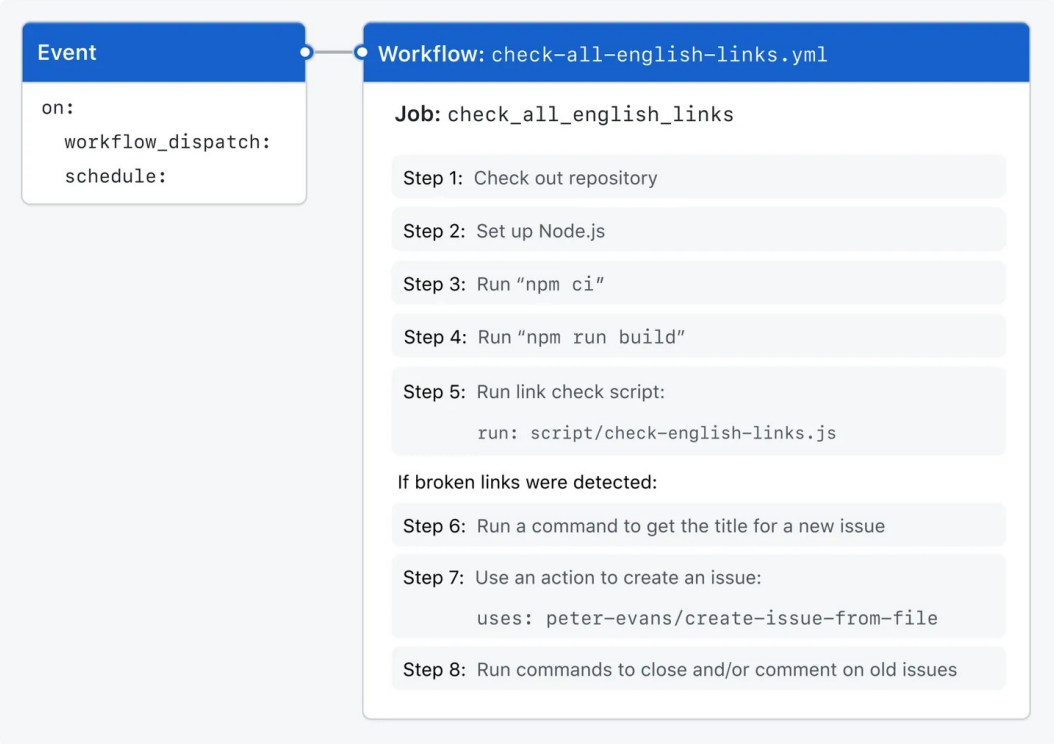
In this article

Example overview
Features used in this example
Example workflow
Next steps

Example overview

This article uses an example workflow to demonstrate some of the main CI features of GitHub Actions. When this workflow is triggered, it automatically runs a script that checks whether the GitHub Docs site has any broken links. If any broken links are found, the workflow uses the GitHub CLI to create a GitHub issue with the details.

The following diagram shows a high level view of the workflow's steps and how they run within the job:



Features used in this example

The example workflow demonstrates the following capabilities of GitHub Actions.

Feature	Implementation
Running a workflow at regular intervals	`schedule`

Setting permissions for the token	`permissions`
Preventing a job from running unless specific conditions are met	`if`
Referencing secrets in a workflow	Secrets
Cloning your repository to the runner	`actions/checkout`
Installing `node` on the runner	`actions/setup-node`
Using a third-party action	`peter-evans/create-issue-from-file`
Running shell commands on the runner	`run`
Running a script on the runner	Using `script/check-english-links.js`
Generating an output file	Piping the output using the `>` operator
Checking for existing issues using GitHub CLI	`gh issue list`
Commenting on an issue using GitHub CLI	`gh issue comment`

Example workflow

The following workflow was created by the GitHub Docs Engineering team. To review the latest version of this file in the `github/docs` repository, see `check-all-english-links.yml`.

The following workflow checks all English links one time per day and reports broken links by creating a new issue for the docs content team to review.

YAML Beside Inline

```yaml
name: Check all English links
```

This defines the name of the workflow as it will appear in the "Actions" tab of the GitHub repository.

```yaml
on:
  workflow_dispatch:
  schedule:
    - cron: '40 19 * * *' # once a day at 19:40 UTC / 11:40 PST
```

Defines the `workflow_dispatch` and `scheduled` as triggers for the workflow.

The `workflow_dispatch` event lets you manually run this workflow from the UI. For more information, see `workflow_dispatch`.

The `schedule` event lets you use `cron` syntax to define a regular interval for automatically triggering the workflow. For more information, see `schedule`.

```yaml
permissions:
  contents: read
  issues: write
```

Modifies the default permissions granted to `GITHUB_TOKEN`. This will vary depending

on the needs of your workflow. For more information, see "Assigning permissions to jobs."

```
jobs:
```

Groups together all the jobs that run in the workflow file.

```
  check_all_english_links:
    name: Check all links
```

Defines a job with the ID `check_all_english_links`, and the name `Check all links`, that is stored within the `jobs` key.

```
    if: github.repository == 'github/docs-internal'
```

Only run the `check_all_english_links` job if the repository is named `docs-internal` and is within the `github` organization. Otherwise, the job is marked as *skipped*.

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

Configures the job to run on an Ubuntu Linux runner. This means that the job will execute on a fresh virtual machine hosted by GitHub. For syntax examples using other runners, see "Workflow syntax for GitHub Actions."

```
    env:
      GITHUB_TOKEN: ${{ secrets.DOCUBOT_READORG_REPO_WORKFLOW_SCOPES }}
      FIRST_RESPONDER_PROJECT: Docs content first responder
      REPORT_AUTHOR: docubot
      REPORT_LABEL: broken link report
      REPORT_REPOSITORY: github/docs-content
```

Creates custom environment variables, and redefines the built-in `GITHUB_TOKEN` variable to use a custom secret. These variables will be referenced later in the workflow.

```
    steps:
```

Groups together all the steps that will run as part of the `check_all_english_links` job. Each job in the workflow has its own `steps` section.

```
      - name: Check out repo's default branch
        uses: actions/checkout@v4
```

The `uses` keyword tells the job to retrieve the action named `actions/checkout`. This is an action that checks out your repository and downloads it to the runner, allowing you to run actions against your code (such as testing tools). You must use the checkout action any time your workflow will run against the repository's code or you are using an action defined in the repository.

```
      - name: Setup Node
        uses: actions/setup-node@v3
```

```
        with:
          node-version: 16.13.x
          cache: npm
```

This step uses the `actions/setup-node` action to install the specified version of the `node` software package on the runner, which gives you access to the `npm` command.

```
      - name: Run the "npm ci" command
        run: npm ci
      - name: Run the "npm run build" command
        run: npm run build
```

The `run` keyword tells the job to execute a command on the runner. In this case, the `npm ci` and `npm run build` commands are run as separate steps to install and build the Node.js application in the repository.

```
      - name: Run script
        run: |
          script/check-english-links.js > broken_links.md
```

This `run` command executes a script that is stored in the repository at `script/check-english-links.js`, and pipes the output to a file called `broken_links.md`.

```
      - if: ${{ failure() }}
        name: Get title for issue
        id: check
        run: echo "title=$(head -1 broken_links.md)" >> $GITHUB_OUTPUT
```

If the `check-english-links.js` script detects broken links and returns a non-zero (failure) exit status, then use a workflow command to set an output that has the value of the first line of the `broken_links.md` file (this is used the next step).

`check-english-links.js` returns 0 if no links are broken, and 1 if any links are broken. When an Actions step's exit code is 1, the action run's job status is failure and the run ends.

The following steps create an issue for the broken link report only if any links are broken, so `if: ${{ failure() }}` ensures the steps run despite the previous step's failure of the job.

```
      - if: ${{ failure() }}
        name: Create issue from file
        id: broken-link-report
        uses: peter-evans/create-issue-from-
file@ceef9be92406ace67ab5421f66570acf213ec395
        with:
          token: ${{ env.GITHUB_TOKEN }}
          title: ${{ steps.check.outputs.title }}
          content-filepath: ./broken_links.md
          repository: ${{ env.REPORT_REPOSITORY }}
          labels: ${{ env.REPORT_LABEL }}
```

Uses the `peter-evans/create-issue-from-file` action to create a new GitHub issue. This example is pinned to a specific version of the action, using the `ceef9be92406ace67ab5421f66570acf213ec395` SHA.

```
      - if: ${{ failure() }}
```

```
        name: Close and/or comment on old issues
        env:
          NEW_REPORT_URL: 'https://github.com/${{ env.REPORT_REPOSITORY
}}/issues/${{ steps.broken-link-report.outputs.issue-number }}'
        run: |
          gh alias set list-reports "issue list \
                                       --repo ${{ env.REPORT_REPOSITORY }} \
                                       --author ${{ env.REPORT_AUTHOR }} \
                                       --label '${{ env.REPORT_LABEL }}'"
          previous_report_url=$(gh list-reports \
                                  --state all \
                                  --limit 2 \
                                  --json url \
                                  --jq '.[].url' \
                                  | grep -v ${{ env.NEW_REPORT_URL }} | head
-1)
```

Uses `gh issue list` to locate the previously created issue from earlier runs. This is aliased to `gh list-reports` for simpler processing in later steps.

```
          gh issue comment ${{ env.NEW_REPORT_URL }} --body "← [Previous
report]($previous_report_url)"
```

`gh issue comment` is used to add a comment to the new issue that links to the previous one.

```
          for issue_url in $(gh list-reports \
                                  --json assignees,url \
                                  --jq '.[] | select (.assignees != []) |
.url'); do
            if [ "$issue_url" != "${{ env.NEW_REPORT_URL }}" ]; then
              gh issue comment $issue_url --body "→ [Newer report](${{
env.NEW_REPORT_URL }})"
            fi
          done
          for issue_url in $(gh list-reports \
                                  --search 'no:assignee' \
                                  --json url \
                                  --jq '.[].url'); do
            if [ "$issue_url" != "${{ env.NEW_REPORT_URL }}" ]; then
              gh issue comment $issue_url --body "→ [Newer report](${{
env.NEW_REPORT_URL }})"
```

If an issue from a previous run is open and assigned to someone, then use `gh issue comment` to add a comment with a link to the new issue without closing the old report. To get the issue URL, the `jq` expression processes the resulting JSON output.

If an issue from a previous run is open and is not assigned to anyone, use `gh issue comment` to add a comment with a link to the new issue. Then use `gh issue close` and `gh issue edit` to close the issue and remove it from the project board.

```
              gh issue close $issue_url
```

Use `gh issue close` to close the old issue.

```
              gh issue edit $issue_url --remove-project "${{
env.FIRST_RESPONDER_PROJECT }}"
            fi
          done
```

Use `gh issue edit` to edit the old issue and remove it from a specific GitHub project board.

Next steps

- To learn about GitHub Actions concepts, see "Understanding GitHub Actions."
- For more step-by-step guide for creating a basic workflow, see "Quickstart for GitHub Actions."
- If you're comfortable with the basics of GitHub Actions, you can learn about workflows and their features at "About workflows."