

Automating Projects using Actions

You can use GitHub Actions to automate your projects.

In this article

GitHub Actions workflows

Example workflow authenticating with a GitHub App

Example workflow authenticating with a personal access token

GitHub Actions workflows

This article demonstrates how to use the GraphQL API and GitHub Actions to add a pull request to an organization project. In the example workflows, when the pull request is marked as "ready for review", a new task is added to the project with a "Status" field set to "Todo", and the current date is added to a custom "Date posted" field.

You can copy one of the workflows below and modify it as described in the table below to meet your needs.

A project can span multiple repositories, but a workflow is specific to a repository. Add the workflow to each repository that you want your project to track. For more information about creating workflow files, see "[Quickstart for GitHub Actions](#)."

This article assumes that you have a basic understanding of GitHub Actions. For more information about GitHub Actions, see "[GitHub Actions documentation](#)."

For more information about other changes you can make to your project through the API, see "[Using the API to manage Projects](#)."

You may also want to use the **actions/add-to-project** workflow, which is maintained by GitHub and will add the current issue or pull request to the project specified. For more information, see the [actions/add-to-project](#) repository and README.

Note: `GITHUB_TOKEN` is scoped to the repository level and cannot access projects. To access projects you can either create a GitHub App (recommended for organization projects) or a personal access token (recommended for user projects). Workflow examples for both approaches are shown below.

Example workflow authenticating with a GitHub App


For more information about authenticating in a GitHub Actions workflow with a GitHub App, see "[Making authenticated API requests with a GitHub App in a GitHub Actions workflow](#)."

- 1 Create a GitHub App or choose an existing GitHub App owned by your organization. For more information, see "[Registering a GitHub App](#)."

- 2 Give your GitHub App read and write permissions to organization projects. For this specific example, your GitHub App will also need read permissions to repository pull requests and repository issues. For more information, see "[Modifying a GitHub App registration](#)."

Note: You can control your app's permission to organization projects and to repository projects. You must give permission to read and write organization projects; permission to read and write repository projects will not be sufficient.

- 3 Install the GitHub App in your organization. Install it for all repositories that your project needs to access. For more information, see "[Installing your own GitHub App](#)."
- 4 Store your GitHub App's ID as a secret in your repository or organization. In the following workflow, replace `APP_ID` with the name of the secret. You can find your app ID on the settings page for your app or through the App API. For more information, see "[Apps](#)."
- 5 Generate a private key for your app. Store the contents of the resulting file as a secret in your repository or organization. (Store the entire contents of the file, including `-----BEGIN RSA PRIVATE KEY-----` and `-----END RSA PRIVATE KEY-----`.) In the following workflow, replace `APP_PEM` with the name of the secret. For more information, see "[Managing private keys for GitHub Apps](#)."
- 6 In the following workflow, replace `YOUR_ORGANIZATION` with the name of your organization. For example, `octo-org`. Replace `YOUR_PROJECT_NUMBER` with your project number. To find the project number, look at the project URL. For example, `https://github.com/orgs/octo-org/projects/5` has a project number of 1. In order for this specific example to work, your project must also have a "Date posted" date field.

YAML Beside Inline 

```
name: Add PR to project
```

```
on:
  pull_request:
    types:
      - ready_for_review
jobs:
  track_pr:
```

This workflow runs whenever a pull request in the repository is marked as "ready for review".

```
- name: Generate token
  id: generate_token
  uses: actions/create-github-app-token@v1
  with:
    app_id: ${ secrets.APP_ID }
    private_key: ${ secrets.APP_PEM }
```

Uses the [actions/create-github-app-token](#) action to generate an installation access token for your app from the app ID and private key. The installation access token is

accessed later in the workflow as `{{ steps.generate_token.outputs.token }}`.

Replace `APP_ID` with the name of the secret that contains your app ID.

Replace `APP_PEM` with the name of the secret that contains your app private key.

```
- name: Get project data
  env:
    GITHUB_TOKEN: {{ steps.generate_token.outputs.token }}
    ORGANIZATION: YOUR_ORGANIZATION
    PROJECT_NUMBER: YOUR_PROJECT_NUMBER
```

Sets environment variables for this step.

Replace `YOUR_ORGANIZATION` with the name of your organization. For example, `octo-org`.

Replace `YOUR_PROJECT_NUMBER` with your project number. To find the project number, look at the project URL. For example, `https://github.com/orgs/octo-org/projects/5` has a project number of 5.

```
run: |
  gh api graphql -f query='
    query($org: String!, $number: Int!) {
      organization(login: $org){
        projectV2(number: $number) {
          id
          fields(first:20) {
            nodes {
              ... on ProjectV2Field {
                id
                name
              }
              ... on ProjectV2SingleSelectField {
                id
                name
                options {
                  id
                  name
                }
              }
            }
          }
        }
      }
    }' -f org=$ORGANIZATION -F number=$PROJECT_NUMBER >
  project_data.json
```

Uses [GitHub CLI](#) to query the API for the ID of the project and return the name and ID of the first 20 fields in the project. `fields` returns a union and the query uses inline fragments (`... on`) to return information about any `ProjectV2Field` and `ProjectV2SingleSelectField` fields. The response is stored in a file called `project_data.json`.

```
echo 'PROJECT_ID='${jq '.data.organization.projectV2.id'
project_data.json} >> $GITHUB_ENV
echo 'DATE_FIELD_ID='${jq
'.data.organization.projectV2.fields.nodes[] | select(.name== "Date posted")
|.id' project_data.json} >> $GITHUB_ENV
echo 'STATUS_FIELD_ID='${jq
'.data.organization.projectV2.fields.nodes[] | select(.name== "Status") |
.id' project_data.json} >> $GITHUB_ENV
echo 'TODO_OPTION_ID='${jq
```

```

'.data.organization.projectV2.fields.nodes[] | select(.name== "Status") |
.options[] | select(.name=="Todo") |.id' project_data.json) >> $GITHUB_ENV

```

Parses the response from the API query and stores the relevant IDs as environment variables. Modify this to get the ID for different fields or options. For example:

- To get the ID of a field called `Team`, add `echo 'TEAM_FIELD_ID=$(jq '.data.organization.projectV2.fields.nodes[] | select(.name== "Team") | .id' project_data.json) >> $GITHUB_ENV`.
- To get the ID of an option called `Octoteam` for the `Team` single select field, add `echo 'OCTOTEAM_OPTION_ID=$(jq '.data.organization.projectV2.fields.nodes[] | select(.name== "Team") |.options[] | select(.name=="Octoteam") |.id' project_data.json) >> $GITHUB_ENV`.

Note: This workflow assumes that you have a project with a single select field called "Status" that includes an option called "Todo" and a date field called "Date posted". You must modify this section to match the fields that are present in your table.

```

- name: Add PR to project
  env:
    GITHUB_TOKEN: ${ steps.generate_token.outputs.token }
    PR_ID: ${ github.event.pull_request.node_id }

```

Sets environment variables for this step. `GITHUB_TOKEN` is the token generated in the first step. `PR_ID` is the ID of the pull request that triggered this workflow.

```

run: |
  item_id="$( gh api graphql -f query='
    mutation($project:ID!, $pr:ID!) {
      addProjectV2ItemById(input: {projectId: $project, contentId:
$pr}) {
        item {
          id
        }
      }
    }' -f project=$PROJECT_ID -f pr=$PR_ID --jq
'.data.addProjectV2ItemById.item.id')"

```

Uses [GitHub CLI](#) and the API to add the pull request that triggered this workflow to the project. The `jq` flag parses the response to get the ID of the created item.

```

echo 'ITEM_ID=$item_id >> $GITHUB_ENV

```

Stores the ID of the created item as an environment variable.

```

- name: Get date
  run: echo "DATE=$(date +"%Y-%m-%d")" >> $GITHUB_ENV

```

Saves the current date as an environment variable in `yyyy-mm-dd` format.

```

- name: Set fields
  env:
    GITHUB_TOKEN: ${ steps.generate_token.outputs.token }

```

Sets environment variables for this step. `GITHUB_TOKEN` is the token generated in the first step.

```

run: |
  gh api graphql -f query='
    mutation (
      $project: ID!
      $item: ID!
      $status_field: ID!
      $status_value: String!
      $date_field: ID!
      $date_value: Date!
    ) {
      set_status: updateProjectV2ItemFieldValue(input: {
        projectId: $project
        itemId: $item
        fieldId: $status_field
        value: {
          singleSelectOptionId: $status_value
        }
      }) {
        projectV2Item {
          id
        }
      }
      set_date_posted: updateProjectV2ItemFieldValue(input: {
        projectId: $project
        itemId: $item
        fieldId: $date_field
        value: {
          date: $date_value
        }
      }) {
        projectV2Item {
          id
        }
      }
    }' -f project=$PROJECT_ID -f item=$ITEM_ID -f
status_field=$STATUS_FIELD_ID -f status_value=${{ env.TODO_OPTION_ID }} -f
date_field=$DATE_FIELD_ID -f date_value=$DATE --silent

```

Sets the value of the `Status` field to `Todo`. Sets the value of the `Date posted` field.

Example workflow authenticating with a personal access token [↗](#)

- 1 Create a personal access token (classic) with the `project` and `repo` scopes. For more information, see "[Managing your personal access tokens](#)."
- 2 Save the personal access token as a secret in your repository or organization.
- 3 In the following workflow, replace `YOUR_TOKEN` with the name of the secret. Replace `YOUR_ORGANIZATION` with the name of your organization. For example, `octo-org`. Replace `YOUR_PROJECT_NUMBER` with your project number. To find the project number, look at the project URL. For example, `https://github.com/orgs/octo-org/projects/5` has a project number of 5.

YAML

Beside

Inline



```

name: Add PR to project
on:
  pull_request:
    types:
      - ready_for_review

```

```

jobs:
  track_pr:
    runs-on: ubuntu-latest
    steps:

```

This workflow runs whenever a pull request in the repository is marked as "ready for review".

```

- name: Get project data
  env:
    GITHUB_TOKEN: ${ secrets.YOUR_TOKEN }}
    ORGANIZATION: YOUR_ORGANIZATION
    PROJECT_NUMBER: YOUR_PROJECT_NUMBER

```

Sets environment variables for this step.

If you are using a personal access token, replace `YOUR_TOKEN` with the name of the secret that contains your personal access token.

Replace `YOUR_ORGANIZATION` with the name of your organization. For example, `octo-org`.

Replace `YOUR_PROJECT_NUMBER` with your project number. To find the project number, look at the project URL. For example, `https://github.com/orgs/octo-org/projects/5` has a project number of 5.

```

run: |
  gh api graphql -f query='
    query($org: String!, $number: Int!) {
      organization(login: $org){
        projectV2(number: $number) {
          id
          fields(first:20) {
            nodes {
              ... on ProjectV2Field {
                id
                name
              }
              ... on ProjectV2SingleSelectField {
                id
                name
                options {
                  id
                  name
                }
              }
            }
          }
        }
      }
    }' -f org=$ORGANIZATION -F number=$PROJECT_NUMBER >
  project_data.json

```

Uses [GitHub CLI](#) to query the API for the ID of the project and return the name and ID of the first 20 fields in the project. `fields` returns a union and the query uses inline fragments (`... on`) to return information about any `ProjectV2Field` and `ProjectV2SingleSelectField` fields. The response is stored in a file called `project_data.json`.

```

    echo 'PROJECT_ID=$(jq '.data.organization.projectV2.id'
  project_data.json) >> $GITHUB_ENV
    echo 'DATE_FIELD_ID=$(jq

```

```

'.data.organization.projectV2.fields.nodes[] | select(.name== "Date posted")
| .id' project_data.json) >> $GITHUB_ENV
echo 'STATUS_FIELD_ID='${jq
'.data.organization.projectV2.fields.nodes[] | select(.name== "Status") |
.id' project_data.json) >> $GITHUB_ENV
echo 'TODO_OPTION_ID='${jq
'.data.organization.projectV2.fields.nodes[] | select(.name== "Status") |
.options[] | select(.name=="Todo") |.id' project_data.json) >> $GITHUB_ENV

```

Parses the response from the API query and stores the relevant IDs as environment variables. Modify this to get the ID for different fields or options. For example:

- To get the ID of a field called `Team`, add `echo 'TEAM_FIELD_ID='${jq`
`'.data.organization.projectV2.fields.nodes[] | select(.name== "Team") | .id'`
`project_data.json) >> $GITHUB_ENV`.
- To get the ID of an option called `Octoteam` for the `Team` single select field, add
`echo 'OCTOTEAM_OPTION_ID='${jq '.data.organization.projectV2.fields.nodes[] |`
`select(.name== "Team") |.options[] | select(.name=="Octoteam") |.id'`
`project_data.json) >> $GITHUB_ENV`.

Note: This workflow assumes that you have a project with a single select field called "Status" that includes an option called "Todo" and a date field called "Date posted". You must modify this section to match the fields that are present in your table.

```

- name: Add PR to project
  env:
    GITHUB_TOKEN: ${ secrets.YOUR_TOKEN }}
    PR_ID: ${ github.event.pull_request.node_id }}

```

Sets environment variables for this step. Replace `YOUR_TOKEN` with the name of the secret that contains your personal access token.

```

run: |
  item_id="$( gh api graphql -f query='
    mutation($project:ID!, $pr:ID!) {
      addProjectV2ItemById(input: {projectId: $project, contentId:
$pr}) {
        item {
          id
        }
      }
    }' -f project=$PROJECT_ID -f pr=$PR_ID --jq
'.data.addProjectV2ItemById.item.id')"

```

Uses [GitHub CLI](#) and the API to add the pull request that triggered this workflow to the project. The `jq` flag parses the response to get the ID of the created item.

```

echo 'ITEM_ID='${item_id >> $GITHUB_ENV

```

Stores the ID of the created item as an environment variable.

```

- name: Get date
  run: echo "DATE=$(date +"%Y-%m-%d")" >> $GITHUB_ENV

```

Saves the current date as an environment variable in `yyyy-mm-dd` format.

```

- name: Set fields

```

```
env:
  GITHUB_TOKEN: ${ secrets.YOUR_TOKEN }}
```

Sets environment variables for this step. Replace `YOUR_TOKEN` with the name of the secret that contains your personal access token.

```
run: |
  gh api graphql -f query='
mutation (
  $project: ID!
  $item: ID!
  $status_field: ID!
  $status_value: String!
  $date_field: ID!
  $date_value: Date!
) {
  set_status: updateProjectV2ItemFieldValue(input: {
    projectId: $project
    itemId: $item
    fieldId: $status_field
    value: {
      singleSelectOptionId: $status_value
    }
  }) {
    projectV2Item {
      id
    }
  }
  set_date_posted: updateProjectV2ItemFieldValue(input: {
    projectId: $project
    itemId: $item
    fieldId: $date_field
    value: {
      date: $date_value
    }
  }) {
    projectV2Item {
      id
    }
  }
}' -f project=$PROJECT_ID -f item=$ITEM_ID -f
status_field=$STATUS_FIELD_ID -f status_value=${ env.TODO_OPTION_ID } -f
date_field=$DATE_FIELD_ID -f date_value=$DATE --silent
```

Sets the value of the `Status` field to `Todo` . Sets the value of the `Date posted` field.

Legal

© 2023 GitHub, Inc. [Terms](#) [Privacy](#) [Status](#) [Pricing](#) [Expert services](#) [Blog](#)