Quickstart for GitHub REST API

In this article

Learn how to get started with the GitHub REST API.

GitHub CLI curl JavaScript

This article describes how to quickly get started with the GitHub REST API using GitHub CLI, JavaScript, or `curl`. For a more detailed guide, see "Getting started with the REST API."

Getting started using GitHub CLI

Using GitHub CLI in the command line

GitHub CLI is the easiest way to use the GitHub REST API from the command line.

1. Install GitHub CLI if you haven't installed it yet. For installation instructions, see the GitHub CLI repository.
2. Use the `auth login` subcommand to authenticate to GitHub CLI. For more information, see the GitHub CLI `auth login` documentation.

```
gh auth login
```

3. Use the `api` subcommand to make your API request. For more information, see the GitHub CLI `api` documentation.

```
gh api repos/octocat/Spoon-Knife/issues
```

Using GitHub CLI in GitHub Actions

You can also use GitHub CLI in your GitHub Actions workflows. For more information, see "Using GitHub CLI in workflows."

Instead of using the `gh auth login` command, pass an access token as an environment variable called `GH_TOKEN`. GitHub recommends that you use the built-in `GITHUB_TOKEN` instead of creating a token. If this is not possible, store your token as a secret and replace `GITHUB_TOKEN` in the example below with the name of your secret. For more information about `GITHUB_TOKEN`, see "Automatic token authentication." For more information about secrets, see "Using secrets in GitHub Actions."

```yaml
on:
  workflow_dispatch:
jobs:
  use_api:
    runs-on: ubuntu-latest
    permissions:
      issues: read
    steps:
      - env:
          GH_TOKEN: ${{ secrets.GITHUB_TOKEN }}
        run: |
          gh api repos/octocat/Spoon-Knife/issues
```

If you are authenticating with a GitHub App, you can create an installation access token within your workflow:

1. Store your GitHub App's ID as a secret. In the following example, 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 API. For more information, see "GitHub Apps" in the REST API documentation. For more information about secrets, see "Using secrets in GitHub Actions."

2. Generate a private key for your app. Store the contents of the resulting file as a secret. (Store the entire contents of the file, including `-----BEGIN RSA PRIVATE KEY-----` and `-----END RSA PRIVATE KEY-----`.) In the following example, replace `APP_PEM` with the name of the secret. For more information, see "Managing private keys for GitHub Apps."

3. Add a step to generate a token, and use that token instead of `GITHUB_TOKEN`. Note that this token will expire after 60 minutes. For example:

   ```yaml
   on:
     workflow_dispatch:
   jobs:
     track_pr:
       runs-on: ubuntu-latest
       steps:
         - name: Generate token
           id: generate_token
           uses: actions/create-github-app-token@v1
           with:
             app_id: ${{ secrets.APP_ID }}
             private_key: ${{ secrets.APP_PEM }}
         - name: Use API
           env:
             GH_TOKEN: ${{ steps.generate_token.outputs.token }}
           run: |
             gh api repos/octocat/Spoon-Knife/issues
   ```

Getting started using JavaScript

You can use Octokit.js to interact with the GitHub REST API in your JavaScript scripts. For more information, see "Scripting with the REST API and JavaScript."

Using Octokit.js

1. Create an access token. For example, create a personal access token or a GitHub App user access token. For more information, see "Creating a personal access

token" or "Identifying and authorizing users for GitHub Apps."

Warning: Treat your access token like a password.

To keep your token secure, you can store your token as a secret and run your script through GitHub Actions. For more information, see the "Using Octokit.js in GitHub Actions" section.

You can also store your token as a Codespaces secret and run your script in Codespaces. For more information, see "Managing encrypted secrets for your codespaces."

If these options are not possible, consider using another CLI service to store your token securely.

2. Install `octokit`. For example, `npm install octokit`. For other ways to install or load `octokit`, see the Octokit.js README.

3. Import `octokit` in your script. For example, `import { Octokit } from "octokit";`. For other ways to import `octokit`, see the Octokit.js README.

4. Create an instance of `Octokit` with your token. Replace `YOUR-TOKEN` with your token.

```
const octokit = new Octokit({
  auth: 'YOUR-TOKEN'
});
```

5. Use `octokit.request` to execute your request. Send the HTTP method and path as the first argument. Specify any path, query, and body parameters in an object as the second argument. For example, in the following request the HTTP method is `GET`, the path is `/repos/{owner}/{repo}/issues`, and the parameters are `owner: "octocat"` and `repo: "Spoon-Knife"`.

```
await octokit.request("GET /repos/{owner}/{repo}/issues", {
  owner: "octocat",
  repo: "Spoon-Knife",
});
```

Using Octokit.js in GitHub Actions

You can also execute your JavaScript scripts in your GitHub Actions workflows. For more information, see "Workflow syntax for GitHub Actions."

GitHub recommends that you use the built-in `GITHUB_TOKEN` instead of creating a token. If this is not possible, store your token as a secret and replace `GITHUB_TOKEN` in the example below with the name of your secret. For more information about `GITHUB_TOKEN`, see "Automatic token authentication." For more information about secrets, see "Using secrets in GitHub Actions."

The following example workflow:

1. Checks out the repository content

2. Sets up Node.js

3. Installs `octokit`

4. Stores the value of `GITHUB_TOKEN` as an environment variable called `TOKEN` and runs

`.github/actions-scripts/use-the-api.mjs`, which can access that environment variable as `process.env.TOKEN`

Example workflow:

```
on:
  workflow_dispatch:
jobs:
  use_api_via_script:
    runs-on: ubuntu-latest
    permissions:
      issues: read
    steps:
      - name: Check out repo content
        uses: actions/checkout@v4

      - name: Setup Node
        uses: actions/setup-node@v3
        with:
          node-version: '16.17.0'
          cache: npm

      - name: Install dependencies
        run: npm install octokit

      - name: Run script
        run: |
          node .github/actions-scripts/use-the-api.mjs
        env:
          TOKEN: ${{ secrets.GITHUB_TOKEN }}
```

Example JavaScript script, with the file path `.github/actions-scripts/use-the-api.mjs`:

```
import { Octokit } from "octokit"

const octokit = new Octokit({
  auth: process.env.TOKEN
});

try {
  const result = await octokit.request("GET /repos/{owner}/{repo}/issues", {
      owner: "octocat",
      repo: "Spoon-Knife",
    });

  const titleAndAuthor = result.data.map(issue => {title: issue.title, authorID:
issue.user.id})

  console.log(titleAndAuthor)

} catch (error) {
  console.log(`Error! Status: ${error.status}. Message:
${error.response.data.message}`)
}
```

If you are authenticating with a GitHub App, you can create an installation access token within your workflow:

1. Store your GitHub App's ID as a secret. In the following example, 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 "GitHub Apps." For more information about secrets, see "Using secrets in GitHub Actions."

2. Generate a private key for your app. Store the contents of the resulting file as a secret. (Store the entire contents of the file, including `-----BEGIN RSA PRIVATE KEY--`

`---` and `-----END RSA PRIVATE KEY-----`.) In the following example, replace `APP_PEM` with the name of the secret. For more information, see "Managing private keys for GitHub Apps."

3. Add a step to generate a token, and use that token instead of `GITHUB_TOKEN`. Note that this token will expire after 60 minutes. For example:

```yaml
on:
  workflow_dispatch:
jobs:
  use_api_via_script:
    runs-on: ubuntu-latest
    steps:
      - name: Check out repo content
        uses: actions/checkout@v4

      - name: Setup Node
        uses: actions/setup-node@v3
        with:
          node-version: '16.17.0'
          cache: npm

      - name: Install dependencies
        run: npm install octokit

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

      - name: Run script
        run: |
          node .github/actions-scripts/use-the-api.mjs
        env:
          TOKEN: ${{ steps.generate_token.outputs.token }}
```

Getting started using `curl`

Using `curl` in the command line

1. Install `curl` if it isn't already installed on your machine. To check if `curl` is installed, execute `curl --version` in the command line. If the output is information about the version of `curl`, it is installed. If you get a message similar to `command not found: curl`, you need to download and install `curl`. For more information, see the curl project download page.

2. Create an access token. For example, create a personal access token or a GitHub App user access token. For more information, see "Creating a personal access token" or "Identifying and authorizing users for GitHub Apps."

   Warning: Treat your access token like a password.

   To keep your token secure, you can store your token as a Codespaces secret and use the command line through Codespaces. For more information, see "Managing encrypted secrets for your codespaces."

   You can also use GitHub CLI instead of `curl`. GitHub CLI will take care of authentication for you. For more information, see the GitHub CLI version of this page.

If these options are not possible, consider using another CLI service to store your token securely.

3. Use the `curl` command to make your request. Pass your token in an `Authorization` header. Replace `YOUR-TOKEN` with your token.

```
curl --request GET \
--url "https://api.github.com/repos/octocat/Spoon-Knife/issues" \
--header "Accept: application/vnd.github+json" \
--header "Authorization: Bearer YOUR-TOKEN"
```

Note: In most cases, you can use `Authorization: Bearer` or `Authorization: token` to pass a token. However, if you are passing a JSON web token (JWT), you must use `Authorization: Bearer`.

Using `curl` commands in GitHub Actions

You can also use `curl` commands in your GitHub Actions workflows.

GitHub recommends that you use the built-in `GITHUB_TOKEN` instead of creating a token. If this is not possible, store your token as a secret and replace `GITHUB_TOKEN` in the example below with the name of your secret. For more information about `GITHUB_TOKEN`, see "Automatic token authentication." For more information about secrets, see "Using secrets in GitHub Actions."

```
on:
  workflow_dispatch:
jobs:
  use_api:
    runs-on: ubuntu-latest
    permissions:
      issues: read
    steps:
      - env:
          GH_TOKEN: ${{ secrets.GITHUB_TOKEN }}
        run: |
          curl --request GET \
          --url "https://api.github.com/repos/octocat/Spoon-Knife/issues" \
          --header "Accept: application/vnd.github+json" \
          --header "Authorization: Bearer $GH_TOKEN"
```

If you are authenticating with a GitHub App, you can create an installation access token within your workflow:

1. Store your GitHub App's ID as a secret. In the following example, 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 "GitHub Apps." For more information about secrets, see "Using secrets in GitHub Actions."

2. Generate a private key for your app. Store the contents of the resulting file as a secret. (Store the entire contents of the file, including `-----BEGIN RSA PRIVATE KEY-----` and `-----END RSA PRIVATE KEY-----`.) In the following example, replace `APP_PEM` with the name of the secret. For more information, see "Managing private keys for GitHub Apps."

3. Add a step to generate a token, and use that token instead of `GITHUB_TOKEN`. Note

that this token will expire after 60 minutes. For example:

```
on:
  workflow_dispatch:
jobs:
  use_api:
    runs-on: ubuntu-latest
    steps:
      - name: Generate token
        id: generate_token
        uses: actions/create-github-app-token@v1
        with:
          app_id: ${{ secrets.APP_ID }}
          private_key: ${{ secrets.APP_PEM }}

      - name: Use API
        env:
          GH_TOKEN: ${{ steps.generate_token.outputs.token }}
        run: |
          curl --request GET \
          --url "https://api.github.com/repos/octocat/Spoon-Knife/issues" \
          --header "Accept: application/vnd.github+json" \
          --header "Authorization: Bearer $GH_TOKEN"
```

Next steps

For a more detailed guide, see "Getting started with the REST API."