Generating a user access token for a GitHub App

In this article

You can generate a user access token for your GitHub App in order to attribute app activity to a user.

About user access tokens

Note: User access tokens that expire are currently an optional feature and are subject to change. To opt in or out of the token expiration feature, see "Activating optional features for GitHub Apps." For more information, see "Expiring user-to-server access tokens for GitHub Apps."

A user access token is a type of OAuth token. Unlike a traditional OAuth token, the user access token does not use scopes. Instead, it uses fine-grained permissions. A user access token only has permissions that both the user and the app have. For example, if the app was granted permission to write the contents of a repository, but the user can only read the contents, then the user access token can only read the contents.

Similarly, a user access token can only access resources that both the user and app can access. For example, if an app is granted access to repository `A` and `B`, and the user can access repository `B` and `C`, the user access token can access repository `B` but not `A` or `C`. You can use the REST API to check which installations and which repositories within an installation a user access token can access. For more information, see `GET /user/installations` and `GET /user/installations/{installation_id}/repositories` in "GitHub App installations."

When you make API requests with a user access token, the rate limits for user access tokens apply. For more information, see "Rate limits for GitHub Apps."

By default, the user access token expires after 8 hours. You can use a refresh token to regenerate a user access token. For more information, see "Refreshing user access tokens."

Users can revoke their authorization of a GitHub App. For more information, see "Token expiration and revocation." If a user revokes their authorization of a GitHub App, the app will receive the `github_app_authorization` webhook. GitHub Apps cannot unsubscribe from this event. If your app receives this webhook, you should stop calling the API on behalf of the user who revoked the token. If your app continues to use a revoked access token, it will receive the `401 Bad Credentials` error. For more information about this webhook, see "Webhook events and payloads."

You should keep user access tokens and refresh tokens secure. For more information, see "Best practices for creating a GitHub App."

Note: If a user reports that they cannot see resources owned by their organization after authorizing your GitHub App and the organization uses SAML SSO, instruct the user to start an active SAML session for their organization before reauthorizing. For more information, see "SAML and GitHub Apps" in the GitHub Enterprise Cloud documentation.

Using the web application flow to generate a user access token

If your app runs in the browser, you should use the web application flow to generate a user access token. For a tutorial about using the web application flow, see "Building a "Login with GitHub" button with a GitHub App."

1. Direct the user to this URL, and add any necessary query parameters from the following list of parameters: `http(s)://HOSTNAME/login/oauth/authorize`. For example, this URL specifies the `client_id` and `state` parameters: `http(s)://HOSTNAME/login/oauth/authorize?client_id=12345&state=abcdefg`.

Query parameter	Type	Description
`client_id`	`string`	**Required.** The client ID for your GitHub App. The client ID is different from the app ID. You can find the client ID on the settings page for your app. For more information about navigating to the settings page for your GitHub App, see "Modifying a GitHub App registration."
`redirect_uri`	`string`	The URL in your application where users will be sent after authorization. This must be an exact match to one of the URLs you provided as a "Callback URL" in your app's settings and can't contain any additional parameters.
`state`	`string`	When specified, the value should contain a random string to protect against forgery attacks, and it can also contain any other arbitrary data.
`login`	`string`	When specified, the web application flow will prompt users with a specific account they can use for signing in and authorizing your app.
`allow_signup`	`boolean`	Whether unauthenticated users will be offered an option to sign up for GitHub during the OAuth flow. The default is `true`. Use `false` when a policy prohibits signups.

2. If the user accepts your authorization request, GitHub will redirect the user to one of the callback URLs in your app settings, and provide a `code` query parameter you can use in the next step to create a user access token. If you specified `redirect_uri` in the previous step, that callback URL will be used. Otherwise, the first callback URL on your app's settings page will be used.

   If you specified the `state` parameter in the previous step, GitHub will also include a `state` parameter. If the `state` parameter does not match the `state` parameter that you sent in the previous step, the request cannot be trusted, and the web application flow should be aborted.

3. Exchange the `code` from the previous step for a user access token by making a `POST` request to this URL, along with the following query parameters:
   `http(s)://HOSTNAME/login/oauth/access_token`

Query parameter	Type	Description
`client_id`	`string`	**Required.** The client ID for your GitHub App. The client ID is different from the app ID. You can find the client ID on the settings page for your app. For more information about navigating to the settings page for your GitHub App, see "Modifying a GitHub App registration."
`client_secret`	`string`	**Required.** The client secret for your GitHub App. You can generate a client secret on the settings page for your app.
`code`	`string`	**Required.** The code you received in the previous step.
`redirect_uri`	`string`	The URL in your application where users will be sent after authorization. This must be an exact match to one of the URLs you provided as a "Callback URL" when setting up your GitHub App and can't contain any additional parameters.
`repository_id`	`string`	The ID of a single repository that the user access token can access. If the GitHub App or user cannot access the repository, this will be ignored. Use this parameter to restrict the access of the user access token further.

4. GitHub will give a response that includes the following parameters:

Response parameter	Type	Description
`access_token`	`string`	The user access token. The token starts with `ghu_`.

`expires_in`	`integer`	The number of seconds until `access_token` expires. If you disabled expiration of user access tokens, this parameter will be omitted. The value will always be `28800` (8 hours).
`refresh_token`	`string`	The refresh token. If you disabled expiration of user access tokens, this parameter will be omitted. The token starts with `ghr_`.
`refresh_token_expires_in`	`integer`	The number of seconds until `refresh_token` expires. If you disabled expiration of user access tokens, this parameter will be omitted. The value will always be `15811200` (6 months).
`scope`	`string`	The scopes that the token has. This value will always be an empty string. Unlike a traditional OAuth token, the user access token is limited to the permissions that both your app and the user have.
`token_type`	`string`	The type of token. The value will always be `bearer`.

5. Use the user access token from the previous step to make API requests on behalf of the user. Include the user access token in the `Authorization` header of an API request. For example:

```
curl --request GET \
--url "http(s)://<em>HOSTNAME</em>/api/v3/user" \
--header "Accept: application/vnd.github+json" \
--header "Authorization: Bearer USER_ACCESS_TOKEN" \
--header "X-GitHub-Api-Version: 2022-11-28"
```

Using the device flow to generate a user access token

Note: The device flow is in public beta and subject to change.

If your app is headless or does not have access to a browser, you should use the device flow to generate a user access token. For example, CLI tools, simple Raspberry Pis, and desktop applications should use the device flow. For a tutorial that uses device flow, see "Building a CLI with a GitHub App."

Before you can use the device flow, you must first enable it in your app's settings. For more information on enabling device flow, see "Modifying a GitHub App registration."

The device flow uses the OAuth 2.0 Device Authorization Grant.

1. Send a `POST` request to `http(s)://HOSTNAME/login/device/code` along with a `client_id` query parameter. The client ID is different from the app ID. You can find the client ID on the settings page for your app. For more information about navigating to the settings page for your GitHub App, see "Modifying a GitHub App registration."

2. GitHub will give a response that includes the following query parameters:

Response parameter	Type	Description
`device_code`	`string`	A verification code that is used to verify the device. This code is 40 characters long.
`user_code`	`string`	A verification code that your application should display so that the user can enter the code in a browser. This code is 8 characters with a hyphen in the middle. For example, `WDJB-MJHT`.
`verification_uri`	`string`	The URL where users need to enter their `user_code`. The URL is: `http(s)://HOSTNAME/login/device`.
`expires_in`	`integer`	The number of seconds before the `device_code` and `user_code` expire. The default is 900 seconds (15 minutes).
`interval`	`integer`	The minimum number of seconds that must pass before you can make a new access token request (`POST http(s)://HOSTNAME/login/oauth/access_token`) to complete the device authorization. If you make a request before this interval passes, then you will hit the rate limit and receive a `slow_down` error. The default is 5 seconds.

3. Prompt the user to enter the `user_code` from the previous step at `http(s)://HOSTNAME/login/device`.

   If the user does not enter the code before the `expires_in` time passes, the code will be invalid. In this case, you should restart the device flow.

4. Poll `POST http(s)://HOSTNAME/login/oauth/access_token` along with the `client_id`, `device_code`, and `grant_type` query parameters (described below) until the device and user codes expire or the user has successfully authorized the app by entering the `user_code`.

Query parameter	Type	Description
`client_id`	`string`	**Required.** The client ID for your GitHub App.

`device_code`	`string`	**Required.** The device verification code you received in the previous step.
`grant_type`	`string`	**Required.** The grant type must be `urn:ietf:params:oauth:grant-type:device_code`.
`repository_id`	`string`	The ID of a single repository that the user access token can access. If the GitHub App or user cannot access the repository, this will be ignored. Use this parameter to restrict the access of the user access token further.

Do not poll this endpoint at a higher frequency than the frequency indicated by `interval`. If you do, you will hit the rate limit and receive a `slow_down` error. The `slow_down` error response adds 5 seconds to the last `interval`.

Until the user enters the code, GitHub will respond with a 200 status and an `error` response query parameter.

Error name	Description
`authorization_pending`	This error occurs when the authorization request is pending and the user hasn't entered the user code yet. The app is expected to keep polling the `POST http(s)://HOSTNAME/login/oauth/access_token` at a frequency no faster than the frequency specified by `interval`.
`slow_down`	When you receive the `slow_down` error, 5 extra seconds are added to the minimum `interval` or timeframe required between your requests using `POST http(s)://HOSTNAME/login/oauth/access_token`. For example, if the starting interval required at least 5 seconds between requests and you get a `slow_down` error response, you must now wait a minimum of 10 seconds before making a new request for a token. The error response includes the new `interval` that you must use.
`expired_token`	If the device code expired, then you will see the `token_expired` error. You must make a new request for a device code.
`unsupported_grant_type`	The grant type must be `urn:ietf:params:oauth:grant-type:device_code` and included as an input parameter when you poll the OAuth token request `POST http(s)://HOSTNAME/login/oauth/access_token`.
`incorrect_client_credentials`	For the device flow, you must pass your app's client ID, which you can find on your app settings page. The client ID is different from the app ID and client secret.
`incorrect_device_code`	The `device_code` provided is not valid.

`access_denied`	When a user clicks cancel during the authorization process, you'll receive an `access_denied` error, and the user won't be able to use the verification code again.
`device_flow_disabled`	Device flow has not been enabled in the app's settings. For more information on enabling device flow, see "Modifying a GitHub App registration."

5. Once the user has entered the `user_code`, GitHub will give a response that includes the following query parameters:

Response parameter	Type	Description
`access_token`	`string`	The user access token. The token starts with `ghu_`.
`expires_in`	`integer`	The number of seconds until `access_token` expires. If you disabled expiration of user access tokens, this parameter will be omitted. The value will always be `28800` (8 hours).
`refresh_token`	`string`	The refresh token. If you disabled expiration of user access tokens, this parameter will be omitted. The token starts with `ghr_`.
`refresh_token_expires_in`	`integer`	The number of seconds until `refresh_token` expires. If you disabled expiration of user access tokens, this parameter will be omitted. The value will always be `15811200` (6 months).
`scope`	`string`	The scopes that the token has. This value will always be an empty string. Unlike a traditional OAuth token, the user access token is limited to the permissions that both your app and the user have.
`token_type`	`string`	The type of token. The value will always be `bearer`.

6. Use the user access token from the previous step to make API requests on behalf of the user. Include the user access token in the `Authorization` header of an API request. For example:

```
curl --request GET \
--url "http(s)://<em>HOSTNAME</em>/api/v3/user" \
--header "Accept: application/vnd.github+json" \
--header "Authorization: Bearer USER_ACCESS_TOKEN" \
--header "X-GitHub-Api-Version: 2022-11-28"
```

Generating a user access token when a user installs your app

If you select **Request user authorization (OAuth) during installation** in your app settings, GitHub will start the web application flow immediately after a user installs your app.

You can generate a user access token with this method regardless of whether the app is installed on a user account or an organization account. However, if the app was installed on an organization account, you will need to use the web application flow or device flow to generate a user access token for other users in the organization.

1. When a user installs your app, GitHub will redirect the user to `http(s)://HOSTNAME/login/oauth/authorize?client_id=CLIENT_ID`, where `CLIENT_ID` is the client ID of your app.

2. If the user accepts your authorization request, GitHub will redirect the user to the first callback URL in your app settings, and provide a `code` query parameter.

   If you want to control which callback URL is used, do not select **Request user authorization (OAuth) during installation**. Instead, direct users through the full web application flow and specify the `redirect_uri` parameter.

3. Exchange the `code` from the previous step for a user access token by making a `POST` request to this URL, along with the following query parameters: `http(s)://HOSTNAME/login/oauth/access_token`

Query parameter	Type	Description
`client_id`	`string`	**Required.** The client ID for your GitHub App. The client ID is different from the app ID. You can find the client ID on the settings page for your app. For more information about navigating to the settings page for your GitHub App, see "Modifying a GitHub App registration."
`client_secret`	`string`	**Required.** The client secret for your GitHub App. You can generate a client secret on the settings page for your app.
`code`	`string`	**Required.** The code you received in the previous step.
`redirect_uri`	`string`	The URL in your application where users will be sent after authorization. This must be an exact match to one of the URLs you provided as a "Callback URL" when setting up your GitHub App and can't contain any additional parameters.
`repository_id`	`string`	The ID of a single repository that the user access token

		can access. If the GitHub App or user cannot access the repository, this will be ignored. Use this parameter to restrict the access of the user access token further.

4. GitHub will give a response that includes the following parameters:

Response parameter	Type	Description
`access_token`	`string`	The user access token. The token starts with `ghu_`.
`expires_in`	`integer`	The number of seconds until `access_token` expires. If you disabled expiration of user access tokens, this parameter will be omitted. The value will always be `28800` (8 hours).
`refresh_token`	`string`	The refresh token. If you disabled expiration of user access tokens, this parameter will be omitted. The token starts with `ghr_`.
`refresh_token_expires_in`	`integer`	The number of seconds until `refresh_token` expires. If you disabled expiration of user access tokens, this parameter will be omitted. The value will always be `15811200` (6 months).
`scope`	`string`	The scopes that the token has. This value will always be an empty string. Unlike a traditional OAuth token, the user access token is limited to the permissions that both your app and the user have.
`token_type`	`string`	The type of token. The value will always be `bearer`.

5. Use the user access token from the previous step to make API requests on behalf of the user. Include the user access token in the `Authorization` header of an API request. For example:

```
curl --request GET \
--url "http(s)://<em>HOSTNAME</em>/api/v3/user" \
--header "Accept: application/vnd.github+json" \
--header "Authorization: Bearer USER_ACCESS_TOKEN" \
--header "X-GitHub-Api-Version: 2022-11-28"
```

Using a refresh token to generate a user access token

By default, user access tokens expires after 8 hours. If you receive a user access token with an expiration, you will also receive a refresh token. The refresh token expire after 6 months. You can use this refresh token to regenerate a user access token. For more information, see "Refreshing user access tokens."

GitHub strongly encourages you to use user access tokens that expire. If you previously opted out of using user access tokens that expire but want to re-enable this feature, see "Activating optional features for GitHub Apps".

Troubleshooting

The following sections outline some errors you may receive when generating a user access token.

Incorrect client credentials

If the `client_id` or `client_secret` that you specify are incorrect, you will receive an `incorrect_client_credentials` error.

To resolve this error, make sure to use the correct credentials for your GitHub App. You can find the client ID and client secret on the settings page for your GitHub App. For more information about navigating to your GitHub App settings page, see "Modifying a GitHub App registration."

Redirect URI mismatch

If you specify a `redirect_uri` that doesn't match one of the callback URLs in your GitHub App registration, you will receive a `redirect_uri_mismatch` error.

To resolve this error, either provide a `redirect_uri` that matches one of the callback URLs for your GitHub App registration, or omit this parameter to default to the first callback URL that is listed on your GitHub App registration. For more information, see "About the user authorization callback URL."

Bad verification code

If you are using device flow and the verification code (`device_code`) that you specified is incorrect, expired, or doesn't match the value that you received from the initial request to `http(s)://HOSTNAME/login/device/code`, you will receive a `bad_verification_code` error.

To resolve this error, you should start the device flow again to get a new code. For more information, see "Using the device flow to generate a user access token."

Bad refresh token

If the refresh token that you specified is invalid or expired, you will receive a `bad_refresh_token` error.

To resolve this error, you must restart the web application flow or device flow to get a new user access token and refresh token. You will only receive a refresh token if your GitHub App has opted in to expiring user access tokens. For more information, see "Refreshing user access tokens."

Unsupported grant type

When you request a user access token via the device flow, the `grant_type` parameter must be `urn:ietf:params:oauth:grant-type:device_code`. When you refresh a user access

token by using a refresh token, the `grant_type` parameter must be `refresh_token`. If you don't use the correct grant type, you will receive an `unsupported_grant_type` error.

Unverified user email

If the user for whom you are trying to generate a user access token has not verified their primary email address with GitHub, you will receive an `unverified_user_email` error.

To resolve this error, prompt the user to verify the primary email address on their GitHub account. For more information, see "Verifying your email address" in the GitHub Free documentation.