Scopes for OAuth apps

In this article

Scopes let you specify exactly what type of access you need. Scopes *limit* access for OAuth tokens. They do not grant any additional permission beyond that which the user already has.

Note: Consider building a GitHub App instead of an OAuth app. GitHub Apps use fine-grained permissions instead of scopes, which give you more control over what your app can do. For more information, see "Differences between GitHub Apps and OAuth apps" and "About creating GitHub Apps."

When setting up an OAuth app on GitHub, requested scopes are displayed to the user on the authorization form.

Note: If you're building a GitHub App, you don't need to provide scopes in your authorization request. For more on this, see "Authenticating with a GitHub App on behalf of a user."

If your OAuth app doesn't have access to a browser, such as a CLI tool, then you don't need to specify a scope for users to authenticate to your app. For more information, see "Authorizing OAuth apps."

Check headers to see what OAuth scopes you have, and what the API action accepts:

```
$ curl -H "Authorization: Bearer OAUTH-TOKEN"
https://api.github.com/users/codertocat -I
HTTP/2 200
X-OAuth-Scopes: repo, user
X-Accepted-OAuth-Scopes: user
```

- `X-OAuth-Scopes` lists the scopes your token has authorized.
- `X-Accepted-OAuth-Scopes` lists the scopes that the action checks for.

Available scopes

Name	Description
`(no scope)`	Grants read-only access to public information (including user profile info, repository info, and gists)
`repo`	Grants full access to public, internal, and private repositories including read and write access to code, commit statuses, repository invitations, collaborators, deployment statuses, and

	repository webhooks. **Note**: In addition to repository related resources, the `repo` scope also grants access to manage organization-owned resources including projects, invitations, team memberships and webhooks. This scope also grants the ability to manage projects owned by users.
`repo:status`	Grants read/write access to commit statuses in public, private, and internal repositories. This scope is only necessary to grant other users or services access to private repository commit statuses *without* granting access to the code.
`repo_deployment`	Grants access to deployment statuses for public and private repositories. This scope is only necessary to grant other users or services access to deployment statuses, *without* granting access to the code.
`public_repo`	Limits access to public repositories. That includes read/write access to code, commit statuses, repository projects, collaborators, and deployment statuses for public repositories and organizations. Also required for starring public repositories.
`repo:invite`	Grants accept/decline abilities for invitations to collaborate on a repository. This scope is only necessary to grant other users or services access to invites *without* granting access to the code.
`security_events`	Grants: read and write access to security events in the code scanning API read and write access to security events in the secret scanning API This scope is only necessary to grant other users or services access to security events *without* granting access to the code.
`admin:repo_hook`	Grants read, write, ping, and delete access to repository hooks in public, private, or internal repositories. The `repo` and `public_repo` scopes grant full access to repositories, including repository hooks. Use the `admin:repo_hook` scope to limit access to only repository hooks.
`write:repo_hook`	Grants read, write, and ping access to hooks in public, private, or internal repositories.
`read:repo_hook`	Grants read and ping access to hooks in public, private, or internal repositories.
`admin:org`	Fully manage the organization and its teams, projects, and memberships.
`write:org`	Read and write access to organization membership, organization projects, and team membership.
`read:org`	Read-only access to organization membership, organization projects, and team membership.
`admin:public_key`	Fully manage public keys

`admin:public_key`	Fully manage public keys.
`write:public_key`	Create, list, and view details for public keys.
`read:public_key`	List and view details for public keys.
`admin:org_hook`	Grants read, write, ping, and delete access to organization hooks. **Note:** OAuth tokens will only be able to perform these actions on organization hooks which were created by the OAuth app. Personal access tokens will only be able to perform these actions on organization hooks created by a user.
`gist`	Grants write access to gists.
`notifications`	Grants: read access to a user's notifications mark as read access to threads watch and unwatch access to a repository, and read, write, and delete access to thread subscriptions.
`user`	Grants read/write access to profile info only. Note that this scope includes `user:email` and `user:follow`.
`read:user`	Grants access to read a user's profile data.
`user:email`	Grants read access to a user's email addresses.
`user:follow`	Grants access to follow or unfollow other users.
`project`	Grants read/write access to user and organization projects.
`read:project`	Grants read only access to user and organization projects.
`delete_repo`	Grants access to delete adminable repositories.
`write:packages`	Grants access to upload or publish a package in GitHub Packages. For more information, see "Publishing a package".
`read:packages`	Grants access to download or install packages from GitHub Packages. For more information, see "Installing a package".
`delete:packages`	Grants access to delete packages from GitHub Packages. For more information, see "Deleting and restoring a package."
`admin:gpg_key`	Fully manage GPG keys.
`write:gpg_key`	Create, list, and view details for GPG keys.
`read:gpg_key`	List and view details for GPG keys.
`codespace`	Grants the ability to create and manage codespaces. Codespaces can expose a GITHUB_TOKEN which may have a different set of scopes. For more information, see "Security in GitHub Codespaces."

`workflow`	Grants the ability to add and update GitHub Actions workflow files. Workflow files can be committed without this scope if the same file (with both the same path and contents) exists on another branch in the same repository. Workflow files can expose `GITHUB_TOKEN` which may have a different set of scopes. For more information, see "Automatic token authentication."
`admin:enterprise`	Gives full control of enterprise functionality. For more information, see "Managing enterprise accounts" in the GraphQL API documentation. Includes `manage_runners:enterprise`, `manage_billing:enterprise`, and `read:enterprise`.
`manage_runners:enterprise`	Gives full control over self-hosted runners within the enterprise. For more information, see "About self-hosted runners."
`manage_billing:enterprise`	Read and write enterprise billing data. For more information, see "Billing" in the REST API documentation.
`read:enterprise`	Read all data on an enterprise profile. Does not include profile data of enterprise members or organizations.
`read:audit_log`	Read audit log data.

Note: Your OAuth app can request the scopes in the initial redirection. You can specify multiple scopes by separating them with a space using `%20`:

```
https://github.com/login/oauth/authorize?
  client_id=...&
  scope=user%20repo_deployment
```

Requested scopes and granted scopes

The `scope` attribute lists scopes attached to the token that were granted by the user. Normally, these scopes will be identical to what you requested. However, users can edit their scopes, effectively granting your application less access than you originally requested. Also, users can edit token scopes after the OAuth flow is completed. You should be aware of this possibility and adjust your application's behavior accordingly.

It's important to handle error cases where a user chooses to grant you less access than you originally requested. For example, applications can warn or otherwise communicate with their users that they will see reduced functionality or be unable to perform some actions.

Also, applications can always send users back through the flow again to get additional permission, but don’t forget that users can always say no.

Check out the Basics of Authentication guide, which provides tips on handling modifiable token scopes.

Normalized scopes

When requesting multiple scopes, the token is saved with a normalized list of scopes, discarding those that are implicitly included by another requested scope. For example, requesting `user,gist,user:email` will result in a token with `user` and `gist` scopes only since the access granted with `user:email` scope is included in the `user` scope.