



Migrating repositories from GitHub.com to GitHub Enterprise Cloud

In this article

About repository migrations with GitHub Enterprise Importer

Prerequisites

- Step 0: Get ready to use the GitHub GraphQL API
- Step 1: Get the ownerld for your migration destination
- Step 2: Identify where you're migrating from
- Step 3: Start your repository migration
- Step 4: Check the status of your migration
- Step 5: Validate your migration and check the error log
- Step 1: Install the GEI extension of the GitHub CLI
- Step 2: Update the GEI extension of the GitHub CLI
- Step 3: Set environment variables
- Step 4: Generate a migration script
- Step 5: Migrate repositories
- Step 6: Validate your migration and check the error log

You can migrate repositories from GitHub.com to GitHub Enterprise Cloud, using the GitHub CLI or the GraphQL API.

API GitHub CLI

About repository migrations with GitHub Enterprise Importer @

You can run your migration with either the GitHub CLI or the API.

The GitHub CLI simplifies the migration process and is recommended for most customers. Advanced customers with heavy customization needs can use the API to build their own integrations with GitHub Enterprise Importer.

To see instructions for using the API, use the tool switcher at the top of the page.

To see instructions for using the GitHub CLI, use the tool switcher at the top of the page.

Prerequisites @

- To ensure you understand the known support limitations of the Importer, review
 "About GitHub Enterprise Importer."
- We strongly recommend that you perform a trial run of your migration and complete your production migration soon after. To learn more about trial run best practices, see "Preparing to run a migration with GitHub Enterprise Importer."
- While not required, we recommend halting work during your production migration.

 The Importer doesn't support delta migrations, so any changes that happen during

the migration will not migrate. If you choose not to halt work during your production migration, you'll need to manually migrate these changes.

• You must be either an organization owner or be granted the migrator role for both the source and destination organizations. For more information, see "Granting the migrator role for GitHub Enterprise Importer."

Step 0: Get ready to use the GitHub GraphQL API

To make GraphQL queries, you'll need to write your own scripts or use an HTTP client like Insomnia.

To learn more about getting started with the GitHub GraphQL API, including how to authenticate, see "Forming calls with GraphQL."

Step 1: Get the ownerId for your migration destination ∂

As an organization owner in GitHub Enterprise Cloud, use the GetOrgInfo query to return the ownerId, also called the organization ID, for the organization you want to own the migrated repositories. You'll need the ownerId to identify your migration destination.

GetOrgInfo query ♂

```
query(
    $login: String!
){
    organization (login: $login)
    {
       login
       id
       name
       databaseId
    }
}
```

Query variable

Description

login

Your organization name.

GetOrgInfo response ∂

```
{
  "data": {
    "organization": {
        "login": "Octo",
        "id": "MDEy0k9yZ2FuaXphdGlvbjU2MTA=",
        "name": "Octo-org",
        "databaseId": 5610
    }
}
```

In this example, MDEy0k9yZ2FuaXphdGlvbjU2MTA= is the organization ID or ownerId, which we'll use in the next step.

Step 2: Identify where you're migrating from ₽

You can set up a migration source using the <code>createMigrationSource</code> query. You'll need to supply the <code>ownerId</code>, or organization ID, gathered from the <code>GetOrgInfo</code> query.

Your migration source is an organization on GitHub.com.

createMigrationSource mutation ∂

```
mutation createMigrationSource($name: String!, $ownerId: ID!) {
   createMigrationSource(input: {name: $name, url: "https://github.com", ownerId:
   $ownerId, type: GITHUB_ARCHIVE}) {
    migrationSource {
      id
      name
      url
      type
   }
   }
}
```

Note: Make sure to use GITHUB_ARCHIVE for type.

Query variable	Description
name	A name for your migration source. This name is for your own reference, so you can use any string.
ownerId	The organization ID of your organization on GitHub Enterprise Cloud.

createMigrationSource response ∂

```
{
  "data": {
    "createMigrationSource": {
        "migrationSource": {
            "id": "MS_kgDaACQxYmYxOWU4Yi0wNzZmLTQ3NTMt0TdkZC1hNGUzZmYxN2U2YzA",
            "name": "GitHub.com Source",
            "url": "https://github.com",
            "type": "GITHUB_SOURCE"
        }
    }
}
```

In this example, $MS_kgDaACQxYmYx0WU4Yi0wNzZmLTQ3NTMt0TdkZC1hNGUzZmYxN2U2YzA$ is the migration source ID, which we'll use in the next step.

Step 3: Start your repository migration ∂

When you start a migration, a single repository and its accompanying data migrates into a brand new GitHub repository that you identify.

If you want to move multiple repositories at once from the same source organization, you can queue multiple migrations. You can run up to 5 repository migrations at the same time.

startRepositoryMigration mutation @

```
mutation startRepositoryMigration (
  $sourceId: ID!,
  $ownerId: ID!,
  $sourceRepositoryUrl: URI!,
  $repositoryName: String!,
  $continueOnError: Boolean!,
  $accessToken: String!,
  $githubPat: String!,
  $targetRepoVisibility: String!
  startRepositoryMigration( input: {
    sourceId: $sourceId,
    ownerId: $ownerId,
    repositoryName: $repositoryName,
    continueOnError: $continueOnError,
    accessToken: $accessToken,
    githubPat: $githubPat,
    targetRepoVisibility: $targetRepoVisibility
    sourceRepositoryUrl: $sourceRepositoryUrl,
  }) {
    repositoryMigration {
      id
      migrationSource {
        id
        name
        type
      }
      sourceUrl
  }
}
```

Query variable	Description
sourceId	Your migration source id returned from the createMigrationSource mutation.
ownerId	The organization ID of your organization on GitHub Enterprise Cloud.
repositoryName	A custom unique repository name not currently used by any of your repositories owned by the organization on GitHub Enterprise Cloud. An error-logging issue will be created in this repository when your migration is complete or has stopped.
continueOnError	Migration setting that allows the migration to continue when encountering errors that don't cause the migration to fail. Must be true or false. We highly recommend setting continueOnError to true so that your migration will continue unless the Importer can't move Git source or the Importer has lost connection and cannot reconnect to complete the migration.
githubPat	The personal access token for your destination organization on GitHub Enterprise Cloud. For personal access token requirements, see "Managing access for GitHub Enterprise Importer."
accessToken	The personal access token for your source.
targetRepoVisibility	The visibility of the new repository. Must be

repository is migrated as private.

SourceRepositoryUrl

The URL of your source repository, using the format

https://github.com/{organization}/{repository}

private, public, or internal. If not set, your

In the next step, you'll use the migration ID returned from the startRepositoryMigration mutation to check the migration status.

Step 4: Check the status of your migration ∂

To detect any migration failures and ensure your migration is working, you can check your migration status using the <code>getMigration</code> query. You can also check the status of multiple migrations with <code>getMigrations</code>.

The <code>getMigration</code> query will return with a status to let you know if the migration is queued , <code>in progress</code> , <code>failed</code> , or <code>completed</code> . If your migration failed, the Importer will provide a reason for the failure.

getMigration query 🔗

```
query (
    $id: ID!
){
    node( id: $id ) {
        ... on Migration {
        id
            sourceUrl
            migrationSource {
                 name
            }
            state
            failureReason
        }
    }
}
```

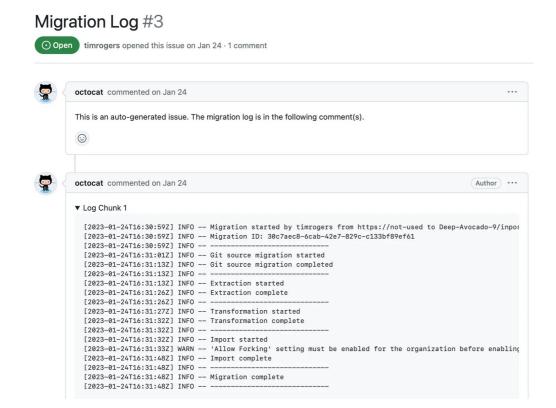
Query variable Description

id

The id of your migration that <u>the</u>
startRepositoryMigration mutation returned.

Step 5: Validate your migration and check the error log $\ensuremath{\mathscr{O}}$

To finish your migration, we recommend that you check the "Migration Log" issue. This issue is created on GitHub in the destination repository.



Finally, we recommend that you review your migrated repositories for a soundness check.

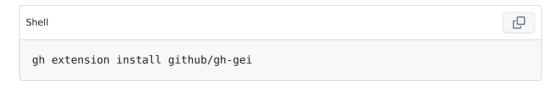
Step 1: Install the GEI extension of the GitHub CLI

If this is your first migration, you'll need to install the GEI extension of the GitHub CLI. For more information about the GitHub CLI, see "About GitHub CLI."

Install the GitHub CLI. For installation instructions for GitHub CLI, see the <u>GitHub CLI repository</u>.

Note: You need version 2.4.0 or newer of GitHub CLI. You can check the version you have installed with the gh --version command.

2 Install the GEI extension.



Any time you need help with the GEI extension, you can use the --help flag with a command. For example, gh gei --help will list all the available commands, and gh gei migrate-repo --help will list all the options available for the migrate-repo command.

Step 2: Update the GEI extension of the GitHub CLI ∂

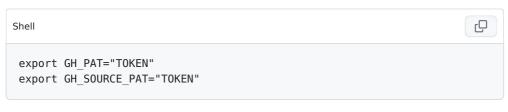
The GEI extension is updated weekly. To make sure you're using the latest version, update the extension.

gh extension upgrade github/gh-gei

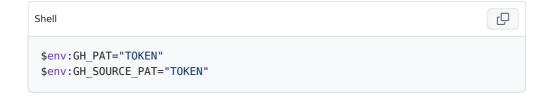
Step 3: Set environment variables &

Before you can use the GEI extension to migrate to GitHub Enterprise Cloud, you must create personal access tokens that can access the source and destination organizations, then set the personal access tokens as environment variables.

- 1 Create and record a personal access token (classic) that will authenticate for the destination organization on GitHub Enterprise Cloud, making sure that the token meets all requirements. For more information, see "Managing access for GitHub Enterprise Importer."
- 2 Create and record a personal access token that will authenticate for the source organization, making sure that this token also meets all of the same requirements.
- 3 Set environment variables for the personal access tokens, replacing TOKEN in the commands below with the personal access tokens you recorded above. Use GH_PAT for the destination organization and GH_SOURCE_PAT for the source organization.
 - If you're using Terminal, use the export command.



• If you're using PowerShell, use the \$env command.



Step 4: Generate a migration script &

If you want to migrate multiple repositories to GitHub Enterprise Cloud at once, use the GitHub CLI to generate a migration script. The resulting script will contain a list of migration commands, one per repository.

Note: Generating a script outputs a PowerShell script. If you're using Terminal, you will need to output the script with the ps1 file extension and install PowerShell for either Mac or Linux to run it.

If you want to migrate a single repository, skip to the next step.

Generating a migration script ₽

To generate a migration script, run the gh gei generate-script command.



If you want the script to download the migration log for each migrated repository, add the --download-migration-logs flag. For more information about migration logs, see "Accessing your migration logs for GitHub Enterprise Importer."

Replace the placeholders in the command above with the following values.

Placeholder	Value
SOURCE	Name of the source organization
DESTINATION	Name of the destination organization
FILENAME	A filename for the resulting migration script
	If you're using Terminal, use a .ps1 file extension as the generated script requires PowerShell to run. You can install PowerShell for Mac or Linux.

Reviewing the migration script &

After you generate the script, review the file and, optionally, edit the script.

- If there are any repositories you don't want to migrate, delete or comment out the corresponding lines.
- If you want any repositories to have a different name in the destination organization, update the value for the corresponding --target-repo flag.

Note: If your repository has more than 10 GB of releases data, releases cannot be migrated. Use the --skip-releases flag to migrate the repository without releases.

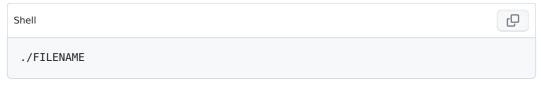
Step 5: Migrate repositories &

You can migrate multiple repositories with a migration script or a single repository with the gh gei migrate-repo command.

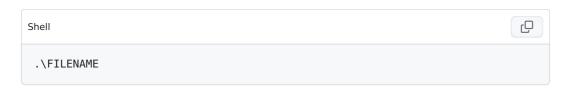
Migrate multiple repositories *∂*

To migrate multiple repositories, run the script you generated above. Replace FILENAME in the commands below with the filename you provided when generating the script.

• If you're using Terminal, use ./.



• If you're using PowerShell, use .\.



To migrate a single repository, use the gh gei migrate-repo command.



Note: If your repository has more than 10 GB of releases data, releases cannot be migrated. Use the --skip-releases flag to migrate the repository without releases.

Replace the placeholders in the command above with the following values.

Placeholder	Value
SOURCE	Name of the source organization
CURRENT-NAME	The name of the repository you want to migrate
DESTINATION	Name of the destination organization
NEW-NAME	The name you want the migrated repository to have

Step 6: Validate your migration and check the error log *∂*

When your migration is complete, we recommend reviewing your migration log. For more information, see "Accessing your migration logs for GitHub Enterprise Importer."

We recommend that you review your migrated repositories for a soundness check.

Legal

© 2023 GitHub, Inc. Terms Privacy Status Pricing Expert services Blog