Troubleshooting your migration with GitHub Enterprise Importer

In this article

If your migration fails or produces unexpected results, you can try common troubleshooting steps.

About troubleshooting steps for GitHub Enterprise Importer

If your migration fails or produces unexpected results, try the first steps for troubleshooting below, which commonly resolve a variety of issues. If these first steps do not resolve your issue, check for error messages in the logs for your migration. Then, locate the error message in this article and try the steps for resolution.

If you're unable to resolve your issue after trying the troubleshooting steps for the error message, you can contact GitHub Support.

First steps for troubleshooting

Before you investigate further, try these troubleshooting steps that commonly resolve a variety of issues.

1. Verify that you're using the latest version of the GitHub CLI extension you're using to migrate. If you're not, upgrade to the latest version.
2. Verify that you meet all the access requirements. For more information, see "Managing access for GitHub Enterprise Importer."
3. Try running the migration again. Some migrations issues are transient, and a second attempt may work.
4. Try running a migration on a different repository with similar data. This will help determine whether the issue is unique to the repository or represents a broader data shape problem.

If these steps do not resolve your issue, review the migration logs for error messages. The log you need to check will depend on whether your migration failed or succeeded.

Troubleshooting failed migrations

If your migration fails, review the verbose log entries produced by the GitHub CLI for each migration. The log file is saved in the same directory where you ran the migration.

The log contains a record of each command you issued and all of the API requests that the GitHub CLI made in response. Failures and error messages normally appear towards the end of the log.

Unable to run migrations

If you see an error like `No access to createMigrationMutation` or `Missing permissions`, your personal account does not have the required access to run the migration. Make sure you're either an organization owner or have been granted the migrator role. For more information about granting the migrator role, see "Migrating repositories with GitHub Enterprise Importer."

> **Note:** If you're migrating between GitHub products, make sure you're an organization owner or have been granted the migrator role for both the source and target organizations.

Resource is protected by organization SAML enforcement

This error indicates that a personal access token you provided to the GitHub CLI needs to be authorized for use with SAML single sign-on. For more information, see "Authorizing a personal access token for use with SAML single sign-on."

`401 Unauthorized` response

Failures that include a `401` status code usually indicate that the personal access token you provided to the GitHub CLI does not have the required scopes. Verify the scopes on the personal access tokens you provided for both your source and destination organizations. For more information about required scopes, see "Managing access for GitHub Enterprise Importer."

`404 Not Found` response

Failures that include a `404` status code usually indicate a typo in one of your commands. Review the migration log for the exact command you entered, and check for typos in the source repository, organization, or project.

`Archive generation failed` response

If you receive a `Archive generation failed...` response when migrating from GitHub Enterprise Server, your repository is probably too large. For more information about

repository size limits, see "Migration support for GitHub Enterprise Importer."

First, try excluding releases from the migration by using the `--skip-releases` flag with the `migrate-repo` command.

If that doesn't work, we'd recommend upgrading to GitHub Enterprise Server 3.8.0 or later. If you're unable to upgrade, another option is to generate your repository archives manually using `ghe-migrator`:

1. Generate a migration archive for your repository. You must only export one repository at a time. For instructions, see "Exporting migration data from your enterprise" in the GitHub Enterprise Server documentation.
2. Upload your migration archive to your choice of blob storage provider.
3. Generate a short-lived URL for your migration archive which is accessible to GitHub.com, such as an AWS S3 pre-signed URL or Azure Blob Storage SAS URL.
4. Call the `migrate-repo` command with the `--git-archive-url` and `--metadata-archive-url` flags both set to the URL of your archive from the previous step.

`cipher name is not supported` error

If you're migrating from Bitbucket Server and receive an error like `cipher name aes256-ctr for openssh key file is not supported` when running a migration, your SSH private key uses an unsupported cipher. For more information about supported ciphers, see "Managing access for GitHub Enterprise Importer."

To generate a new, compatible SSH keypair, run the following command:

Shell

```
ssh-keygen -t ed25519 -Z aes256-cbc -C "your_email@example.com"
```

After generating a new SSH keypair, before you can use the key, you must add the public key to your Bitbucket Server instance's `authorized_keys`.

`Subsystem 'sftp' could not be executed` error

If you're migrating from Bitbucket Server and receive an error like `Subsystem 'sftp' could not be executed`, SFTP is not enabled on your server or your user account does not have SFTP access.

You should contact your server administrator and ask them to enable SFTP access for your user account.

`Source export archive... does not exist` error

If you're migrating from Bitbucket Server and you receive an error like `Source export archive (/var/atlassian/application-data/bitbucket/shared/migration/export/Bitbucket_export_1.tar) does not exist`, the GitHub CLI is looking for your migration archive in the wrong place on your Bitbucket Server instance.

To resolve this issue, set the `--bbs-shared-home` argument for `gh bbs2gh migrate-repo` to your Bitbucket Server or Data Center's shared home directory. The default shared home directory is `/var/atlassian/application-data/bitbucket/shared`, but your configuration may be different.

You can identify the shared home directory in Bitbucket Server.

1. Navigate to the Administration area of your Bitbucket Server or Data Center instance.
2. In the sidebar, under "System," click "Storage."
3. Under "Shared directory," view the location of your server's shared home directory.

If you're running Bitbucket Data Center in cluster mode with multiple notes, your shared directory will be shared between cluster nodes and should be mounted in the same location on each node.

`Repository rule violations found` error

If you receive a `Repository rule violations found` error, such as `GH013: Repository rule violations found for refs/heads/main`, data in the origin repository conflicts with rulesets configured on the destination organization. For more information, see "About rulesets."

You can temporarily disable your rulesets during your migration, or you can use bypass mode or the bypass list to exempt your migration from configured rules. For more information, see "Managing rulesets for repositories in your organization."

`Your push would publish a private email address` error

If you receive a `Git source migration failed` error with `GH007: Your push would publish a private email address`, the Git source you're trying to migrate includes commits authored by an email address that you have blocked from being pushed to GitHub. For more information, see "Blocking command line pushes that expose your personal email address."

To resolve this error, you can either rewrite the Git history to remove the email address, or you can disable the "Block command line pushes that expose my email" setting.

Troubleshooting successful migrations

If your migration succeeds but produces unexpected results, review the migration log for error messages. For more information, see "Accessing your migration logs for GitHub Enterprise Importer."

> **Note:** If the "Migration Log" issue includes "Migration completed" at the bottom, the repository was migrated. Error messages only indicate that specific items within the repository, such as a comment on a pull request, may not have migrated correctly.

- Repository metadata too big to migrate
- Comment not in diff
- Pull request review thread not migrated in pull request
- Team references are broken after an organization migration

Repository metadata too big to migrate

If you see "Repository metadata too big to migrate" in the "Migration Log" issue or the GitHub CLI, your repository exceeds the maximum archive size of 10 GB. This is often caused by large release assets. Try excluding releases from the migration with the `--skip-releases` flag for the `migrate-repo` command.

Comment not in diff

If you're migrating from Azure DevOps, pull request comments on lines that were never changed in the pull request cannot be migrated to GitHub. You'll see this warning for every comment that cannot be migrated for this reason.

> **Note:** Only comments on lines that weren't changed in a pull request are affected by this limitation. Comments on lines that were changed in a pull request are migrated.

Be aware that the affected comments will not be in the migrated repository, but these warnings do not require further action from you.

Pull request review thread not migrated in pull request

This warning is a more generic form of "Comment not in diff." If you see this warning, a comment on a file in a pull request couldn’t be migrated for a different reason than the one described above. Most often, the comment was on a line that was changed at one point in the pull request history, but then the pull request changed so that this was no longer the case.

- The comment is on a file that was deleted later in the pull request history.
- The comment was on code that was changed at one point in the pull request history, but the author later decided to remove the change.
- The pull request was squash merged into a single commit, which prevents GitHub from being able to properly construct the pull request history to properly place the comment.

This problem most frequently impacts closed pull requests.

Be aware that the affected comments will not be in the migrated repository, but these warnings do not require further action from you.

Team references are broken after an organization migration

References to teams, such as `@octo-org/octo-team`, are **not** updated as part of an organization migration. This might cause problems in the destination organization, such as `CODEOWNERS` files not working as expected.

You can either update these references after the migration, or you can preserve your team names by renaming the source organization so that you can use the original name for your destination organization.

For example, if your source organization is `@octo-org`, and your `CODEOWNERS` file includes a reference to the team `@octo-org/octo-team`, you could rename the source organization to `@octo-org-temp` before your migration, allowing you to use `@octo-org` as the name of the new organization. Then, the migrated team would be called `@octo-org/octo-team`, and the `CODEOWNERS` file in the migrated repository will work as expected.

Locked repositories

After a migration, you may find that your source or destination repositories are locked, disabling access to the repository's code and all of its resources, such as issues and pull requests. For more information about locked repositories, see "About locked repositories."

The process for unlocking a repository depends on the GitHub product where the repository is stored.

- If the locked repository is on GitHub Enterprise Server, a site administrator can unlock the repository using the site admin dashboard. For more information, see "Locking a repository" in the GitHub Enterprise Server documentation.
- If the locked repository is on GitHub.com, you can contact us through the GitHub Support portal to unlock the repository.

Note: If your migration failed, not all of your data was migrated. If you choose to unlock and use the repository, there will be data loss. Deleting the locked repository and retrying the migration may be a better option.

Contacting GitHub Support

If you're still unable to resolve your issue after trying the troubleshooting steps above, you can contact GitHub Support through the GitHub Support portal.