Planning your migration to GitHub

In this article

Learn how to plan and execute a successful migration to GitHub or between GitHub products.

About migrations

If you're moving between GitHub products, such as from GitHub Enterprise Server to GitHub Enterprise Cloud, or from another code hosting platform, such as Bitbucket Server or GitLab, to GitHub, you'll want to bring your work with you: your code, the code's history, and all of your past conversations and collaboration.

This guide will usher you through planning and executing a successful migration. You'll learn how to prepare for a migration, the tools that are available to move your data, and how to make your move a success.

Migration terminology

Before using this guide to plan your migration, learn these important terms.

Term	Definition
Code hosting platform	The online tool that you use to host your source code repositories and to collaborate, such as GitHub Enterprise Cloud, GitHub Enterprise Server, Bitbucket Server, and GitLab.com.
Version control system (VCS)	The tool that you use to track and manage changes to your source code on the machine where you're making those changes. For example, if you're using GitHub or GitLab as your code hosting platform, you're using the Git version control system. If you're using Azure DevOps as your code hosting platform, you could be using either Git or Team Foundation Version Control (TFVC) as the underlying version control system. It's also possible that you aren't

	using a VCS at all.
Migration origin	The place you're migrating from. Usually, this will be a code hosting platform, but it might be your own machine or a shared network drive.
Migration destination	The GitHub product that you're moving to.
Migration path	The combination of your migration origin and migration destination, such as "Bitbucket Server to GitHub Enterprise Cloud." For certain migration paths, GitHub offers specialist tools, such as GitHub Enterprise Importer, to help you migrate.

Defining your migration scope

Before you can plan your migration, you need to understand what you want to migrate, and when.

Defining your origin and destination

First, determine where you need to move data from. This is usually, but not always, a code hosting platform.

Your code hosting platform might be a GitHub product, such as GitHub.com or GitHub Enterprise Server, or it might be another code hosting platform, such as Bitbucket Server, GitLab, or Azure DevOps. Depending on the size and complexity of your business, you might be using multiple different code hosting platforms.

If you're not using a code hosting platform at all, you might be storing your code on a shared network drive, for example.

Wherever your code lives, that's your "migration origin."

You'll also need to know which GitHub product you're migrating to, or your "migration destination." This could be GitHub.com, which includes GitHub Enterprise Cloud, or GitHub Enterprise Server.

Building a basic inventory of the repositories you want to migrate

After you've identified your migration origin and destination, establish what data you need to migrate.

You should build a migration inventory with a list of all of the repositories in your migration origin(s) that you need to migrate. We recommend using a spreadsheet. As a starting point, you should record the following data for each repository:

- Name
- Owner: in GitHub, this would be an organization, but in other tools, there might be a different kind of owner
- URL
- Last updated timestamp
- Number of pull requests (or equivalent in your migration origin)
- Number of issues (or equivalent in your migration origin)

If you're migrating from GitHub Enterprise Cloud or GitHub Enterprise Server, you can obtain this data with the `gh-repo-stats` extension for the GitHub CLI. With just a few

commands, `gh-repo-stats` will connect with your migration origin's API and create a CSV with all of the recommended fields. For more information, see the mona-actions/gh-repo-stats repository.

> **Note:** `gh-repo-stats` is a third-party open-source tool which is not supported by GitHub Support. If you need help with this tool, open an issue in its repository.

If you're migrating from Azure DevOps, we recommend the `inventory-report` command in the ADO2GH extension of the GitHub CLI. The `inventory-report` command will connect with the Azure DevOps API, then build a simple CSV with some of the fields suggested above. For more information about how to install the ADO2GH extension of the GitHub CLI, see "Migrating repositories from Azure DevOps to GitHub Enterprise Cloud."

If you're migrating from Bitbucket Server or Bitbucket Data Center, we recommend the `inventory-report` command in the BBS2GH extension of the GitHub CLI. The `inventory-report` command will use your Bitbucket instance's API to build a simple CSV. For more information about how to install the BBS2GH extension of the GitHub CLI, see "Migrating repositories from Bitbucket Server to GitHub Enterprise Cloud."

For other migration origins, create your migration inventory yourself. You could build the spreadsheet using the origin's reporting tools, if available, or API, or you could create the inventory manually.

Whatever approach you choose for your migration inventory, make a note of the process you followed or commands you ran. It's very likely that you'll want to re-run your inventory as you continue to plan your migration.

After you have a list of all of your repositories, you can decide which ones you want to migrate. One option is to migrate absolutely everything. However, a migration is a great opportunity to evaluate your repositories and remove any that are no longer needed. We find that many business have hundreds or even thousands of unused and unneeded repositories, and archiving them can make your migration much simpler.

Measuring the sizes of your repositories

After you've completed your basic migration inventory, collect information about the size of your repositories. If your repositories are large or contain individual files over 100MB, this can make your migration longer and riskier and limit the migration tools that are available to you.

If you're using Git as your version control system, it's not only large files currently in the repository that matter; large files in your repository's history matter too. For example, if you had a file larger than 100MB in your repository in the past, then that file will still be present in your Git history, unless you've rewritten the history to remove all traces of the file. For more information about rewriting history, see "About large files on GitHub."

If you used `gh-repo-stats` to build your inventory, you'll already have some basic information on how big your repositories are. To build a complete migration inventory, you'll need to obtain finer details about the data inside your repositories.

Next, follow the instructions below to add the following data to your migration inventory for each repository:

- The size of the largest file (also known as a "blob")
- The total size of all files ("blobs")

If you're using a version control system other than Git, or your files aren't tracked with a version control system at all, first move the repositories to Git. For more information, see "Source code and history migrations."

Then, use the open-source tool, `git-sizer`, to get this data for your repository.

Prerequisites

1. Install `git-sizer`. For more information, see the github/git-sizer repository.
2. To verify that that `git-sizer` is installed, run `git-sizer –version`. If you see output like `git-sizer release 1.5.0`, installation was successful.
3. Install `jq`. For more information, see Download jq in the `jq` documentation.
4. To verify that `jq` is installed, run `jq –-version`. If you see output like `jq-1.6`, installation was successful.

Measuring repository size with `git-sizer`

1. To clone your repository from the migration origin, run `git clone --mirror`.
2. Navigate to the directory where you cloned your repository.
3. To get the size of the largest file in your repository in bytes, run `git-sizer --no-progress -j | jq ".max_blob_size"`.
4. To get the total size of all files in your repository in bytes, run `git-sizer --no-progress -j | jq “.unique_blob_size”`.
5. Add the values from the previous steps to your inventory.

About migration types

There are three approaches you can take when running a migration, which provide different levels of migration fidelity.

Migration type	Definition	Requirements
Source snapshot	Migrate the current state of your code, as it is today, but don’t include any of the revision history.	Possible for every origin and destination, even if your code isn’t currently tracked in a version control system (VCS).
Source and history	Migrate the current state of your code and its revision history.	Possible if you've been tracking your changes in Git, or a version control system which can be converted to Git before the migration.
Source, history, and metadata	Migrate the current state of your code and its revision history, plus your collaboration history, such as issues and pull requests, and settings.	Requires specialist tools, which are not available for all migration paths.

When deciding what type of migration to complete, consider your organization’s needs and the tools that are available.

You may want to use different strategies for different repositories. For example, you may have some old, archived repositories where the history is not important, while a high-

fidelity migration is critical for your most active code.

About our different migration support models

You may choose to complete a "self-serve migration," where you plan and run your own migration using our documentation only, without any professional support from GitHub.

Alternatively, you may prefer to work with GitHub's Expert Services team or a GitHub Partner, which we call an "expert-led migration." With an expert-led migration, you benefit from the knowledge and experience of an expert who has previously run tens or even hundreds of migrations, and you get access to additional migration tools that aren't available for self-serve.

	Self-serve	Expert-led
Access to documentation	✓	✓
Access to GitHub's full range of tools	×	✓
Topics covered by support	• Execution • Troubleshooting	• Planning • Execution • Troubleshooting
Cost	Free of charge	Contact Expert Services for details

To learn more about expert-led migrations, contact your account representative or Expert Services.

Deciding what tools to use

To plan for your migration, consider the destination and source. These considerations determine the path for your migration. For more information, see "Migration paths to GitHub."

Designing your organization structure for the migration destination

In GitHub, each repository belongs to an organization. Organizations are shared accounts where businesses and open-source projects can collaborate across many projects at once, with sophisticated security and administrative features. For more information, see "About organizations."

Whether you're adopting GitHub for the first time or already using GitHub, pause to consider the most effective structure for your organizations and repositories after your migration. The design you choose can maximize collaboration and discovery and minimize administrative burden, or it can create unnecessary silos and administrative overhead.

We recommend that you minimize the number of organizations and structure them according to one of five archetypes. For detailed guidance, see "Best practices for structuring organizations in your enterprise."

Performing a dry run migration for every repository

Before you continue planning, perform a dry run migration including all of your repositories. Comprehensive dry runs allow you to:

- Verify that the tool you've chosen works for your repositories
- Confirm that the tool meets your requirements
- Understand exactly what data is migrated, and what data is not migrated
- Understand how long your migration will take, to help you schedule your production migration

There's nothing unique about a dry run migration. Just run a normal migration, then delete the repository in the migration destination.

Planning your pre-migration and post-migration steps

Migrating your repositories is only one step in a larger migration process. There will be other steps you need to take, and possibly data or settings you'll need to migrate manually.

The full list of steps required for your migration will depend on your unique circumstances, but there are some pre-migration steps that apply to all migrations:

- Let your users know ahead of time about the upcoming migration and its timeline
- Send reminders shortly before the migration takes place
- Set up user accounts in GitHub for your team
- Send instructions to your users for updating their local repositories to point to your new system

There are also post-migration steps that apply to all migrations:

- Let your users know that the migration is finished
- Link activity to users in your migration destination
- Decommission your migration origin

Here are some other steps you should consider when planning your migration.

Migrating continuous integration (CI) and continuous delivery (CD)

If you're moving between GitHub products, are already using GitHub Actions for CI/CD, and will continue to use GitHub Actions, there's not much to do. The workflow files in your repositories will automatically be migrated for you. If you use self-hosted runners, you will need to set these up in your new GitHub organization, so they're ready to run your workflows.

If you're not using GitHub Actions, the situation is more complicated. If you plan to continue using the same CI/CD provider, you'll need to check that the provider is compatible with GitHub, and connect the provider to your new organization and repositories.

If you're planning to switch to GitHub Actions, we do not recommend doing so at the same time that you migrate your repositories. Instead, wait until a later date, and perform your CI/CD migration as a separate step. This makes the migration process more manageable. When you're ready to migrate, see "Migrating to GitHub Actions."

Migrating integrations

You're likely to be using integrations with your code hosting provider, either developed in-house or provided by other vendors.

If you’re already using GitHub, then you'll need to reconfigure your integrations to point to your new organizations and repositories. If the integration is provided by a vendor, contact the vendor for instructions. If the integration was developed in-house, reconfigure the integration in your new organization, generating new tokens and keys.

If you’re new to GitHub, check whether your integrations are compatible with GitHub, then reconfigure them. If you use integrations that were developed in-house, re-write them to work with the GitHub API. For more information, see "GitHub REST API documentation."

Linking activity to users in your migration destination

If you’re migrating collaboration history, or metadata, as well as code, you’ll need to link users' activity to their new identities in your migration destination.

For example, suppose @octocat created an issue on your GitHub Enterprise Server instance, and you’re moving to GitHub Enterprise Cloud. In GitHub Enterprise Cloud, @octocat might have a completely different username. The attribution process allows you to link user activity with these new identities.

The way that attribution works differs between tools:

- If you’re using `ghe-migrator`, `gl-exporter`, or `bbs-exporter`, you will decide how you want to attribute data ahead of time and include a mapping file when you import your data.
- If you’re using GitHub Enterprise Importer, data will be linked to placeholder identities called “mannequins”, and you can assign this history to real users after your data is migrated. For more information, see "Reclaiming mannequins for GitHub Enterprise Importer."

Managing teams and permissions

Most customers use teams to manage access to repositories. With teams, instead of giving Mona access to a repository directly, you can add Mona to the Engineering team, and give everyone in the Engineering team access to the repository. For more information, see "About teams."

You can create your teams and add team members before you migrate your repositories. You may want to manage your members through your identity provider (IdP) by linking your teams to IdP groups. For more information, see "Synchronizing a team with an identity provider group."

However, you can’t attach your teams to repositories until after you've migrated the repositories.