MIGRATION PLAYBOOK · GITLAB

How to migrate from GitLab to GitHub

Move GitLab projects to GitHub Enterprise Cloud with GitHub Enterprise Importer, keeping history, issues and merge requests, then convert GitLab CI to GitHub Actions and rebuild permissions.

Updated 4 Nov 2026 7 phases 21 detailed steps 11 official sources

START HERE

Your migration plan

This moves GitLab projects (code, history, wiki, issues and merge requests) to GitHub Enterprise Cloud. GitHub Enterprise Importer (GEI) copies each project in a single run while work on it is frozen, then you finish what it does not move: people, large files, permissions and CI pipelines. Trial runs into a test organisation show how long each project takes, and converting GitLab CI to GitHub Actions is its own piece of work.

RECOMMENDED ROUTE

GitHub Enterprise Importer with the gh gl2gh extension

Moves issues and merge requests as well as code, and can script whole GitLab groups at once.

▼ Other options and when to use them

- **GitHub Importer or a mirror push (code only)**
  : You only need code and history, have a handful of projects, or are moving to GitHub Enterprise Server.
- **GraphQL API migration**
  : A platform team wants to build its own orchestration around the importer.

TOOLS FOR THIS PLAN

Are you ready?
22-question readiness check

Plan your dates
Dated countdown you can add to your calendar

New to the jargon?
Plain-English migration glossary

ORDER TO MIGRATE IN, AND WHY

Set up people and access first, prove the method in a trial, then move projects in waves, finishing each wave's people, permissions and CI before developers switch. GitLab is retired last, once every pipeline runs on GitHub.

1. **Inventory, sizes and the GitLab-to-GitHub mapping**
   Oversized and LFS projects need extra work, and the mapping decides which organisation and team each project lands in.
2. **GitHub accounts for every developer, and the migration tokens**
   Mannequins (placeholder users) can only be reclaimed to accounts that exist, and every run needs working tokens.
3. **Shrink oversized projects (Git LFS or history clean-up)**
   GEI refuses repositories and files over its limits. Rewriting history before the move means everyone re-clones once, not twice.
4. **Trial into a sandbox organisation**
   Shows timings and problems without stopping anyone's work.
5. **Production waves of projects that build together**
   There is no top-up copy, so each wave is frozen while it runs, and projects that depend on each other must switch together.
6. **Per wave: mannequins, LFS objects, teams, rulesets and secrets**
   Without these, history shows placeholders, builds fail and the default branch accepts direct pushes.
7. **CI conversion per project**
   Workflows can only be tested once the code is on GitHub. Keep GitLab runners until each workflow is green.
8. **Developers switch remotes, then GitLab is retired**
   Deleting GitLab cannot be undone and takes pipeline history, artifacts and snippets with it, so it goes last.

SETTINGS CHECKLIST: DECIDE THESE BEFORE YOU COPY

Built-in and third-party tools treat these differently. Tick each one off once you have decided.

Download this checklist (Excel)

- [ ] **Which GitHub you are moving to**
  Whether the destination is GitHub Enterprise Cloud (GitHub's hosted service) or GitHub Enterprise Server (a copy you run yourself).

  ▼ How each tool handles it

Built-in method	GEI: migrates to GitHub Enterprise Cloud only.
Mirror push (git push --mirror)	Works with any GitHub, including Enterprise Server. A mirrored clone includes all branches and tags. GitHub's steps do not cover issues, merge requests or the wiki, so treat them as not copied.

Tip: If you need issues and merge requests on Enterprise Server, check GitHub's current options before you plan the waves.

GitHub Docs: Understand migrations from GitLab

- [ ] **Repository and file size**

Which projects need large files moved to Git LFS or removed from history before the move.

▼ **How each tool handles it**

Built-in method	GEI: 40 GiB of source code per repository (public preview) and 400 MiB per file during migration. After migration GitHub limits files to 100 MiB, and single commits and pushes to 2 GiB. GitLab.com limits its export archive, including metadata, to 40 GB.
Git LFS (git lfs migrate)	Moves large files already in history into Git LFS, or git filter-repo removes them. Both rewrite history, so every clone must be re-cloned.

Tip: Run git-sizer on mirror clones of the biggest projects before you set wave dates.

GitHub Docs: Understand migrations from GitLab

- [ ] **Git LFS files**

Which projects store large files in Git LFS (large files kept outside Git), and who pushes them after each wave.

▼ **How each tool handles it**

Built-in method	GEI: LFS pointer files migrate with the history, but the LFS objects do not. Builds fail until you push them.
Mirror push (git push --mirror)	Also leaves LFS objects behind. GitHub's steps fetch them with git lfs fetch --all and push them with git lfs push --all as separate commands.

Tip: Find LFS projects by looking for filter=lfs in .gitattributes, and add the push to each wave's runsheet.

GitHub Docs: Understand migrations from GitLab

- [ ] **Issues, merge requests and comments**

  Which open merge requests to merge or close before the freeze, and whether flattened comment threads are acceptable.

  ▼ How each tool handles it

Built-in method	GEI: issues and comments migrate, and threaded discussions become flat comments that keep the thread context. Merge requests become pull requests with reviewers, approvers and state events. Comments become review comments only when diff data is in the export, which holds only the latest diff.
Mirror push (git push --mirror)	Copies Git data only. Issues and merge requests stay in GitLab.

  Tip: In the trial, ask each owner to open a few busy merge requests on both sides and compare them.

  GitHub Docs: Understand migrations from GitLab

- [ ] **Labels, milestones, releases and attachments**

  Which labels and release assets must exist on day one.

  ▼ How each tool handles it

Built-in method	GEI: milestones, releases and release assets, uploads (attachments), timeline events and emoji reactions migrate. Labels are not listed as migrated.

  Tip: Compare labels after the trial and recreate any that are missing, before the production waves.

  GitHub Docs: Understand migrations from GitLab

- [ ] **Wiki, snippets and issue boards**

  What to do with snippets, boards and time-tracking data, which have no GitHub copy.

  ▼ How each tool handles it

Built-in method	GEI: the project wiki migrates. Snippets, issue boards, time-tracking data, design-management data and vulnerability reports do not.

  Tip: Export snippets you need to a repository, and rebuild boards as GitHub Projects if teams rely on them.

  GitHub Docs: Understand migrations from GitLab

- [ ] **People and who wrote what**

  Which GitHub account each GitLab user becomes, and when mannequins are reclaimed.

  ▼ **How each tool handles it**

Built-in method	GEI: project members arrive as mannequins (placeholder users). Content stays attributed to the mannequin, and may not show in searches for the real person, until you reclaim it. On self-managed GitLab, a token from a non-administrator gives incomplete exports and loses user attribution.

  Tip: Make sure every developer has a GitHub account in your enterprise before the first wave, and reclaim mannequins straight after each one.

  GitHub Docs: Reclaiming mannequins for GitHub Enterprise Importer

- [ ] **Group membership and permissions**

  Which GitHub team each GitLab group becomes, and which role each team gets on each repository.

  ▼ **How each tool handles it**

Built-in method	GEI: group settings and group membership do not migrate. You create the teams, add members and grant access yourself.

  Tip: Export group members from GitLab before the waves, so you can rebuild teams the same day.

  GitHub Docs: Understand migrations from GitLab

- [ ] **Branch protection, approvals and webhooks**

  Which protections, approval rules and integrations each repository needs on day one.

  ▼ **How each tool handles it**

Built-in method	GEI: repository policies such as required approvals, merge trains, pipeline gates and mirroring do not migrate. Webhooks are not in the GitLab export.

  Tip: Recreate protections as GitHub rulesets straight after each wave, before developers switch, or the default branch accepts direct pushes.

  GitHub Docs: Understand migrations from GitLab

- [ ] **CI/CD pipelines and variables**

When each project's pipelines are converted to GitHub Actions, and where secret values come from.

▼ **How each tool handles it**

Built-in method	GEI: pipelines and pipeline schedules do not migrate. CI/CD variables, job logs and artifacts, and pipeline triggers are not in the GitLab export at all.
GitHub Actions Importer	Converts GitLab pipelines to workflows and opens a pull request with them. Masked variable values and artifact reports need manual work, and caching between workflows is not supported.

Tip: Run the Actions Importer audit before you set dates, and take secret values from your secret store, not from GitLab.

GitHub Docs: Understand migrations from GitLab

- [ ] **Freeze and top-up copies**

How long each wave is frozen, and how projects are made read-only during it.

▼ **How each tool handles it**

Built-in method	GEI: no delta migrations (a top-up copy of only what changed). Anything pushed, commented or merged in GitLab during or after the run does not reach GitHub and must be moved by hand.

Tip: Archive the projects, or reduce developers to read-only, at the start of each freeze.

GitHub Docs: Migrate your repositories from GitLab

- [ ] **Repository visibility**

Whether each new repository should be private, internal (visible to your enterprise) or public.

▼ **How each tool handles it**

Built-in method	GEI (gh gl2gh migrate-repo): new repositories are private unless you set --target-repo-visibility to public or internal.

Tip: Record the visibility for each project in your mapping sheet, so nothing becomes public by mistake.

GitHub Docs: Migrate your repositories from GitLab

- [ ] **Tokens and access**

Who runs the migration, and which tokens and roles they need.

▼ How each tool handles it

Built-in method	GEI: needs a GitHub classic personal access token (fine-grained tokens are not accepted) with repo, workflow and admin:org scopes (read:org for someone with the migrator role), authorised for single sign-on. The GitLab token needs api and read_repository scopes, and on self-managed GitLab it must belong to an administrator, with project export enabled.

Tip: Set the tokens to expire just after your last wave.

GitHub Docs: Manage access for a GitLab migration

☐ **Organisation rulesets**

Whether rulesets that check commits are switched on before or after the migration.

▼ How each tool handles it

Built-in method	GEI: organisation rulesets, such as commit author email rules, can make a migration fail when old commits break them.

Tip: Migrate into an organisation without blocking rulesets, or exclude the new repositories, and apply the rulesets afterwards.

GitHub Docs: Understand migrations from GitLab

THE PLAN, PHASE BY PHASE

1 Plan the move — Before you start

List every project with its size, open merge requests, LFS and CI use,

and owner.

Decide which GitHub organisation and team each GitLab group maps to, with a naming rule for nested paths.

Make sure every developer has a GitHub account in your enterprise.

Group projects that build together into waves, each with

a trial date and a freeze window.

> ⚠ **Watch out:** GEI stops at 40 GiB of source code per repository and 400 MiB per file during migration. GitLab.com also limits its export archive, which includes issues and other metadata, to 40 GB.
>
> **WAYS AROUND IT**
>
> - Run git-sizer on mirror clones of the biggest projects, and give oversized ones their own wave.
>
> **TOOLS THAT CAN DO IT**
>
> **Git LFS (git lfs migrate)**
> Moves large files already in history into Git LFS, or git filter-repo removes them. Both rewrite history, so every clone must be re-cloned.

> ⚠ **Watch out:** GEI only migrates to GitHub Enterprise Cloud, not GitHub Enterprise Server.
>
> **WAYS AROUND IT**
>
> - For Enterprise Server, use a mirror push (git clone --mirror then git push --mirror), which moves code and history only.

Show the detailed steps (3) →

2 Set up access

Create a GitHub classic personal access token (GEI does not accept fine-

grained tokens) with repo, workflow and admin:org (read:org for a migrator), and authorise it for single sign-on.

Create a GitLab token with api and read_repository scopes.

Ask an organisation owner to grant you the migrator role if you are not

an owner.

Install the GitHub CLI 2.4.0 or later with the GL2GH extension, plus PowerShell 7 on macOS or Linux.

> **Watch out:** On self-managed GitLab, a token from a non-administrator gives incomplete exports and loses who wrote what.
>
> **WAYS AROUND IT**
>
> - Use an administrator's token, and check project export is enabled before the trial.

> **Watch out:** Organisation rulesets, such as commit author email rules, can make imports fail on old commits.
>
> **WAYS AROUND IT**
>
> - Migrate into an organisation without blocking rulesets, or exclude the migrated repositories, then apply the rulesets afterwards.

Show the detailed steps (3) →

3 Run a trial

Create a sandbox organisation

(a throwaway test organisation such as contoso-sandbox).

Migrate a mix of small, large and busy projects into it. Work does not need to stop.

With each owner, compare branches, tags, issues, pull requests, releases and the wiki with GitLab.

Record how long each took, fix problems in GitLab, then delete the sandbox.

> ⚠ **Watch out:** GitHub does not list labels among the things GEI migrates.
>
> **WAYS AROUND IT**
>
> - Check labels after the trial and recreate any that are missing.

Show the detailed steps (3) →

4 Migrate production

In each wave's freeze window

Merge or close open merge requests you do not want carried across.

At the start of the freeze, archive

the projects or reduce developers to read-only.

Generate the migration script for the group and remove projects outside this wave.

Run the script, keep the output, and download each repository's migration log.

> **Watch out:** GEI has no delta migration (a top-up copy of only what changed), so anything changed in GitLab during the run does not reach GitHub.
>
> **WAYS AROUND IT**
>
> - Freeze each wave's projects for the whole run, and tell developers to push nothing until they switch.

Show the detailed steps (3) →

5 Finish people and permissions

Reclaim mannequins (placeholder users) so history moves to each person's GitHub account.

Push Git LFS objects (large files stored outside Git) to each new repository that uses them.

Create teams, add members and give each team its

role on its repositories.

Recreate branch protections as rulesets, plus webhooks, apps and Actions secrets.

> **Watch out:** GEI migrates only LFS pointer files, so builds fail until the objects are pushed.
>
> **WAYS AROUND IT**
>
> - Mirror-clone from GitLab, run git lfs fetch --all, then git lfs push --all to GitHub.

> **Watch out:** GEI does not move group membership, branch protections, approval rules, webhooks, snippets or issue boards.
>
> **WAYS AROUND IT**
>
> - Export a list per project from GitLab and recreate them straight after each wave, before developers switch.

> **Watch out:** GitLab masked variable values are not exported.
>
> **WAYS AROUND IT**
>
> - Take the values from your secret store when you create Actions secrets.

Show the detailed steps (4) →

6 Convert CI

Install GitHub Actions Importer (it

runs in Docker) and configure it for GitLab.

Run the audit and forecast for the namespace, then a dry-run per project.

Run migrate to open a pull request with the converted workflow.

Fix the manual items with the project

owner, and merge when a run is green.

> **Watch out:** GEI does not move pipelines, CI/CD variables, job artifacts or pipeline triggers.
>
> **WAYS AROUND IT**
>
> - Run the audit before you set dates, and keep GitLab runners until each workflow is green.
>
> **TOOLS THAT CAN DO IT**
>
> **GitHub Actions Importer**
> Converts GitLab pipelines to workflows. Masked variables and artifact reports still need manual work, and caching between workflows is not supported.

Show the detailed steps (3) →

7 Switch and retire GitLab

Ask developers to change their remote with git remote set-url, rather than recloning.

Update IDE integrations, deploy

keys and scripts that clone from GitLab.

Keep migrated GitLab projects archived for an agreed period so old links still work.

Export what your records policy requires, then decommission GitLab.

> **Watch out:** Deleting GitLab projects or cancelling GitLab cannot be undone, and pipeline history, job artifacts, snippets and boards go with it.
>
> **WAYS AROUND IT**
>
> - Keep GitLab read-only until every pipeline runs in Actions and the decommission checklist is signed off.
> - To roll back before then, unarchive GitLab and push back any commits made on GitHub.

Turn this phase into dated tasks with the cutover planner

Show the detailed steps (2) →

You are finished when

- Branch, tag and commit counts match, and the default branch head is the same commit on both sides.
- Issue and pull request counts match, and no current staff remain as unclaimed mannequins.
- LFS files download with their content, and each workflow runs green in Actions.
- Direct pushes to the default branch are rejected, and nothing has been pushed to GitLab since the freeze.

Tools named in this plan are independent third-party products, not endorsed by the platform vendor and not paid placements. Check the tool's own documentation and run a pilot first.

CLICK BY CLICK

Detailed steps

Open a part when you reach it in the plan. Each step says where to go, what to do and what you should see.

About this migration What it covers, limits at a glance —

GitHub Enterprise Importer (GEI) now migrates GitLab projects to GitHub Enterprise Cloud with the `gh gl2gh` extension. It exports each project from GitLab, uploads the archive to blob storage and imports code, wiki, issues, merge requests (as pull requests), milestones and releases. It does not move pipelines, so GitLab CI is converted separately with GitHub Actions Importer.
The order of work: plan the organisation structure and trial runs, set up access and the CLI, migrate a sandbox, migrate production with work halted, then do the follow-up tasks (people, LFS, permissions, CI). If you only need code and history, GitHub Importer or a mirror push is simpler. For the same job from Azure DevOps, see the Azure DevOps to GitHub guide.

NATIVE TOOL	SOURCE	DESTINATION

GitHub Enterprise Importer with the GL2GH extension of the GitHub CLI (gh gl2gh), generally available since August 2026

GitLab.com and currently maintained GitLab Self-Managed versions

GitHub Enterprise Cloud on GitHub.com or GHE.com; GitHub Enterprise Server is not supported

MIGRATED

Git history and wiki, issues, merge requests as pull requests, milestones, releases, comments

NOT MIGRATED

CI/CD pipelines, Git LFS objects, group membership, repository policies, webhooks, CI/CD variables

SIZE LIMIT

40 GiB per repository (public preview); 400 MiB per file during migration

1 What you need before step one

Need	Details	Where to get it
GitHub Enterprise Cloud	An enterprise account on GitHub.com or GHE.com with the destination organisation. GEI does not migrate GitLab to GitHub Enterprise Server.	GitHub sales or your partner; check current pricing
GitHub role	Organisation owner, or a user granted the migrator role by an owner.	GitHub organisation settings
GitHub token	Personal access token (classic), not fine-grained. Owner: repo, workflow, admin:org. Migrator: repo, workflow, read:org.	GitHub > Settings > Developer settings > Personal access tokens > Tokens (classic)
GitLab token	Personal access token with api and read_repository. On self-managed GitLab, an administrator's token.	GitLab > your avatar > Edit profile > Access tokens
GitHub CLI	Version 2.4.0 or later, with the GL2GH extension.	https://cli.github.com
PowerShell	The generated migration script is PowerShell, so macOS and Linux need PowerShell 7 installed.	https://learn.microsoft.com/powershell/scripting/install/installing-powershell

Need	Details	Where to get it
Docker (for CI)	GitHub Actions Importer runs in a container.	https://www.docker.com
IP allow list	If your organisation uses GitHub's IP allow list, add GEI's ranges as listed by GitHub.	GitHub Docs: Manage access (GitLab)

2 Plan the structure, waves and trial

3 steps

GitLab	Moves with GEI?	What to do
Git history, branches, tags, wiki	Yes	Check size limits first
Issues and comments, milestones	Yes	Issue boards do not move
Merge requests	Yes, as pull requests with comments, reviewers, approvers and state	Close stale merge requests first
Releases and assets, uploads, reactions	Yes	Spot-check after the trial
Members	As mannequins	Reclaim after each wave
Labels	Not listed by GitHub as migrated	Check after the trial and recreate if missing
Pipelines, schedules, CI/CD variables	No	GitHub Actions Importer and Actions secrets
Git LFS objects	No (pointers only)	Push with git lfs after migration

GitLab	Moves with GEI?	What to do
Groups, group membership, protections, webhooks, snippets	No	Recreate as teams, rulesets and webhooks

1

Inventory the GitLab projects

Where: Terminal or PowerShell on your migration machine, with the GitHub CLI and the GL2GH extension installed, or the GitLab web UI for each group

List every project with its group path, size, open merge requests, whether it uses LFS and CI, and its owner. The GL2GH extension has an `inventory-report` command in recent releases; run `gh gl2gh inventory-report --help` for its options. Record the results in your Application inventory and dependency map.

✓ **You should see:** One row per project with size, LFS and CI flags, owner and a target GitHub organisation and name.

2

Map groups to organisations and teams

Where: Your Identity and licence mapping sheet

Decide which GitHub organisation each top-level GitLab group lands in, and which GitHub team replaces each subgroup or permission set. Nested GitLab paths become flat repository names, so agree a naming rule (for example `payments-api` for `platform/payments/payments-api`).

✓ **You should see:** Every project has a target organisation, repository name and owning team.

3

Set the waves and freeze windows

Where: Your Migration wave plan

Group projects that build together into the same wave. Give each wave a trial date, a production date with a freeze window, and a CI conversion owner.

✓ **You should see:** The wave plan is signed off by each team lead.

3 Set up access and the CLI

3 steps

1

Create the tokens

Where: GitHub > Settings > Developer settings > Personal access tokens > Tokens (classic), and GitLab > Edit profile > Access tokens

Create a GitHub classic token with `repo`, `workflow` and `admin:org` (organisation owner) or `repo`, `workflow` and `read:org` (migrator). If your organisation uses SAML single sign-on, authorise the token for it. Create a GitLab token with `api` and `read_repository`, as an administrator on self-managed GitLab.

✓ **You should see:** Two tokens stored in your password manager, each with an expiry after your last wave.

2

Grant the migrator role if needed

Where: Terminal or PowerShell on your migration machine, with the GitHub CLI and the GL2GH extension installed, signed in with an organisation owner's token

If the person running the migration is not an organisation owner, an owner grants them the migrator role. The GL2GH extension includes a `grant-migrator-role` command; run `gh gl2gh grant-migrator-role --help` for its arguments, or follow GitHub's "Grant the migrator role" page.

✓ **You should see:** The migrator can run migrations into the destination organisation.

3

Install the GL2GH extension and set the tokens

Where: Terminal or PowerShell on your migration machine, with the GitHub CLI and the GL2GH extension installed

Check `gh --version` is 2.4.0 or later, install the extension, then set the tokens as environment variables. For GHE.com (data residency), also set `TARGET_API_URL` to `https://api.SUBDOMAIN.ghe.com`.

```
gh extension install github/gh-gl2gh
gh extension upgrade github/gh-gl2gh

# Terminal (macOS or Linux)
export GH_PAT="ghp_your_github_classic_token"
export GITLAB_PAT="glpat-your_gitlab_token"

# PowerShell
$env:GH_PAT="ghp_your_github_classic_token"
$env:GITLAB_PAT="glpat-your_gitlab_token"
```

✓ **You should see:** gh gl2gh --help lists migrate-repo and generate-script.

4 Migrate a trial into a sandbox organisation

3 steps −

1

Create the sandbox organisation

Where: GitHub > your enterprise > Organizations > New organization

Create an organisation such as `contoso-sandbox` without blocking rulesets.

✓ **You should see:** The sandbox organisation exists and you are an owner.

2

Migrate one project

Where: Terminal or PowerShell on your migration machine, with the GitHub CLI and the GL2GH extension installed

Run `migrate-repo` with the full group path (for subgroups, `parent-group/subgroup`). `--use-github-storage` uses GitHub-owned blob storage; if you use your own storage, follow GitHub's "Configure blob storage" page. Add `--target-repo-visibility` to set `private` (default), `internal` or `public`. Use `--no-ssl-verify` only for a self-signed certificate on GitLab.

```
gh gl2gh migrate-repo \
  --gitlab-server-url https://gitlab.contoso.com \
  --gitlab-group platform/payments \
  --gitlab-project payments-api \
  --github-org contoso-sandbox \
  --github-repo payments-api \
  --use-github-storage
```

✓ **You should see:** The command reports the migration succeeded, and contoso-sandbox/payments-api shows the code, wiki, issues and pull requests.

3

Check the trial with the team

Where: The sandbox repository on GitHub, with the project owner

Compare branch and tag counts, open and closed issues, merge requests (now pull requests), releases and the wiki with GitLab. Check labels, which GitHub does not list as migrated. Record the elapsed time for wave sizing, and fix anything in GitLab (oversized files, broken refs) before production.

✓ **You should see:** The owner signs off the trial in your Pilot test plan, with any gaps listed.

5 Migrate production repositories

3 steps

1

Freeze the source projects

Where: GitLab > each project > Settings > General > Advanced > Archive project (or reduce developers to read-only access)

At the start of the freeze window, make the wave's projects read-only so nothing changes during the run. Tell developers the freeze has started.

✓ **You should see:** Pushes and new merge requests are refused on the frozen projects.

2

Generate the migration script

Where: Terminal or PowerShell on your migration machine, with the GitHub CLI and the GL2GH extension installed

Generate a script for the group. By default it includes every project the token can see; narrow it with `--gitlab-group`, or `--gitlab-group` plus `--gitlab-project` for one project. Open the script, delete or comment out projects outside this wave, and edit `--github-repo` and `--target-repo-visibility` to match your naming rule.

```
gh gl2gh generate-script \
  --gitlab-server-url https://gitlab.contoso.com \
  --gitlab-group platform \
  --github-org contoso \
  --output migrate-platform.ps1 \
  --use-github-storage

# Review and edit the script, then run it
./migrate-platform.ps1      # Terminal (needs PowerShell 7 installed as pwsh)
.\migrate-platform.ps1     # PowerShell
```

✓ **You should see:** migrate-platform.ps1 lists one migrate-repo line per project in the wave, with the target names you chose.

3

Run the script and read the logs

Where: Terminal or PowerShell on your migration machine, with the GitHub CLI and the GL2GH extension installed

Run the script and keep the console output. Download the migration log for each repository (the extension has a `download-logs` command; see `gh gl2gh download-logs --help`) and look for warnings about skipped items.

✓ **You should see:** Every project in the wave reports success, and the logs show no unexpected skipped items.

6 **Finish people, LFS and permissions** 4 steps

1

Reclaim mannequins

Where: Terminal or PowerShell on your migration machine, with the GitHub CLI and the GL2GH extension installed, signed in with an organisation owner's token

Generate a CSV of mannequins, fill in each person's GitHub username, and reclaim them in bulk. By default each person gets an email invitation and attribution moves when they accept. The commands follow the same pattern as the other GEI extensions; check `gh gl2gh generate-mannequin-csv --help` for the exact arguments.

```
gh gl2gh generate-mannequin-csv --github-org contoso --output mannequins.csv
gh gl2gh reclaim-mannequin --github-org contoso --csv mannequins.csv
```

✓ **You should see:** People accept the invitations and their GitLab history shows under their GitHub accounts.

2

Push Git LFS objects

Where: Terminal on your migration machine with Git LFS installed (https://git-lfs.com)

For each project that uses LFS, mirror-clone it from GitLab, fetch every LFS object and push them to the new GitHub repository.

```
git clone --mirror https://gitlab.contoso.com/platform/payments/payments-api.git
cd payments-api.git
git lfs fetch --all
git lfs push --all https://github.com/contoso/payments-api.git
```

✓ **You should see:** git lfs ls-files in a fresh clone from GitHub lists the files, and they download with content rather than pointers.

3

Create teams and grant access

Where: GitHub > your organisation > Teams > New team, then each repository > Settings > Collaborators and teams

Create the teams from your mapping sheet, add members, and give each team its role (Read, Triage, Write, Maintain or Admin) on its repositories. Link teams to identity provider groups if your enterprise uses team synchronisation.

✓ **You should see:** Each developer can push to the repositories their GitLab role allowed, and no further.

4

Recreate protections, webhooks and secrets

Where: GitHub > each repository (or organisation) > Settings > Rules, Webhooks, and Secrets and variables > Actions

Recreate protected branches and approval rules as rulesets, add the webhooks and apps your GitLab projects used, and add CI/CD variables as Actions secrets or variables. GitLab masked variable values are not exported, so take them from your secret store.

✓ **You should see:** The default branch rejects direct pushes, webhooks deliver test events, and secrets are listed for each repository.

7 Convert GitLab CI to GitHub Actions

3 steps −

1

Install and configure GitHub Actions Importer

Where: Terminal or PowerShell on your migration machine, with the GitHub CLI and the GL2GH extension installed, with Docker running

Install the extension, update the container and run `configure`. Choose GitLab, enter a GitHub classic token with the `workflow` scope, a GitLab token with `read_api`, and your GitLab URL.

```
gh extension install github/gh-actions-importer
gh actions-importer update
gh actions-importer configure
```

✓ **You should see:** The configure prompts finish and gh actions-importer -h lists audit, forecast, dry-run and migrate.

2

Audit, forecast and dry-run

Where: Terminal or PowerShell on your migration machine, with the GitHub CLI and the GL2GH extension installed

Run the audit for the namespace to see which jobs convert fully or partly, the forecast to estimate runner usage, and a dry-run per project to review the generated workflow. `script`, `stages`, `needs`, `image`, `variables` and schedules convert; `rules`, `include`, `services` and `parallel` are partial; masked variables, artifact reports and cross-workflow caching are manual.

```
gh actions-importer audit gitlab --output-dir tmp/audit --namespace platform
gh actions-importer forecast gitlab --output-dir tmp/forecast --namespace platform
gh actions-importer dry-run gitlab --output-dir tmp/dry-run --namespace platform --project
payments-api
```

✓ **You should see:** tmp/audit holds an audit summary, and tmp/dry-run holds a workflow YAML per project for review.

3

Open the workflow pull request

Where: Terminal or PowerShell on your migration machine, with the GitHub CLI and the GL2GH extension installed

Run `migrate` for each project. It opens a pull request with the converted workflow in the GitHub repository. Review it with the project owner, fix the manual items, and merge when a run is green.

```
gh actions-importer migrate gitlab --target-url https://github.com/contoso/payments-api --output-dir tmp/migrate --namespace platform --project payments-api
```

✓ **You should see:** A pull request with the workflow appears in the repository, and its Actions run passes.

8 Switch developers and retire GitLab

2 steps

1

Point local clones at GitHub

Where: Each developer's terminal, inside their existing clone

Change the remote URL rather than recloning, then fetch to confirm. Update IDE integrations, deploy keys and any scripts that clone from GitLab.

```
git remote set-url origin https://github.com/contoso/payments-api.git
git fetch origin
git remote -v
```

✓ **You should see:** git remote -v shows the github.com URL and git fetch succeeds.

2

Keep GitLab read-only, then retire it

Where: GitLab admin area, and your Decommission checklist

Leave migrated projects archived for an agreed period so links in tickets and documents still resolve. Keep GitLab runners until every pipeline runs in Actions. Then export what your records policy requires, and decommission or cancel GitLab.

✓ **You should see:** The decommission checklist is signed off and no build or deploy still references GitLab.

WORTH KNOWING

Expert notes

The detail behind the plan: options compared, and what experienced engineers check first. Tap any item.

Compare all the ways to do this — 3 options −

GitHub Enterprise Importer (gh gl2gh)

MEDIUM EFFORT

Best for: Organisations moving to GitHub Enterprise Cloud that want issues, merge requests and history

+ Migrates code, wiki, issues, merge requests, milestones and releases
+ Scriptable across whole groups with generate-script
– GitHub Enterprise Cloud only, not GitHub Enterprise Server
– No delta: work halts during each run
– CI, LFS objects and group membership need follow-up work

Read more →

GitHub Importer or mirror push (code only)

LOW EFFORT

Best for: A few projects, a GitHub Enterprise Server destination, or teams that keep issues elsewhere

+ Simple: keeps full Git history, branches and tags
+ Works for any GitHub product
– No issues, merge requests, wiki pages or releases
– Still needs LFS, CI and permission work

Read more →

GraphQL API migration

HIGH EFFORT

Best for: Platform teams building their own orchestration around GEI

+ Full control over export, upload and import
+ Same importer as the CLI
– You write and maintain the tooling
– Archives over 100 MiB need your own blob storage

Read more →

OUR ADVICE

Use GitHub Enterprise Importer with `gh gl2gh` when you are moving to GitHub Enterprise Cloud and want issues and merge requests to come with the code. Use

a mirror push for a handful of projects or for GitHub Enterprise Server, and the GraphQL API only if you need your own orchestration. Whichever you choose, plan GitLab CI conversion with GitHub Actions Importer as its own workstream.

Run a trial into a sandbox organisation first

REDUCES RISK

GitHub recommends trial runs to see whether each repository migrates, whether it can be returned to a workable state, and how long it takes. Trial runs do not need work to stop.

WHAT TO DO

Create a test organisation with a name ending in `-sandbox`, migrate a mix of small, large and busy projects, and time them. Use the timings to size the production waves, then delete the sandbox.

GitHub Docs: Migrate your repositories from GitLab

Halt work during the production run, there is no delta

REDUCES RISK

GitHub says the importer does not support delta migrations: anything pushed, commented or merged in GitLab during the migration does not reach GitHub.

WHAT TO DO

Agree a freeze window per wave, merge or close open merge requests you do not want carried across, and make the GitLab projects read-only for the window (Step 5). Developers push nothing to GitLab until they switch remotes.

GitHub Docs: Migrate your repositories from GitLab

Find large repositories and files before the waves

SAVES TIME

GEI is limited to 40 GiB per repository (public preview) and 400 MiB per file during migration; GitHub limits files to 100 MiB after migration, commits and pushes to 2 GiB, and refs to 255 bytes. GitLab.com limits the export archive to 40 GB including metadata.

WHAT TO DO

Run `git-sizer` (https://github.com/github/git-sizer) on a mirror clone of the biggest projects. Move large binaries to Git LFS or clean history before the production run, and give oversized projects their own wave.

TOOLS THAT CAN GET AROUND THIS

Rewriting history to move large files into Git LFS, or remove them, brings a project under GEI's 400 MiB per-file limit before the production run.

Git LFS (git lfs migrate)

GitHub documents `git lfs migrate` to move large files already in history into Git LFS, or `git filter-repo` to remove them. Both rewrite history, so every clone must be re-cloned.

Third-party tools, not endorsed by the platform vendor. Confirm in the tool's documentation and run a pilot first.

GitHub Docs: Understand migrations from GitLab

Push Git LFS objects separately

REDUCES RISK

LFS pointer files migrate, but the binary objects do not. A repository that looks complete fails its first build when the LFS files are missing.

WHAT TO DO

List projects that use LFS (look for `.gitattributes` with `filter=lfs`). After each migration, fetch and push the LFS objects with the commands in Step 6.

GitHub Docs: Understand migrations from GitLab

Use an administrator token on self-managed GitLab

REDUCES RISK

On self-managed instances the GitLab token must belong to an administrator so exports are complete and user attribution is kept. Projects must also be enabled for export.

WHAT TO DO

Create the GitLab personal access token as an instance administrator with the `api` and `read_repository` scopes, and check project export is enabled in your instance's import and export settings before the trial.

GitHub Docs: Manage access for a GitLab migration

Plan people: mannequins and teams

USER IMPACT

Project members arrive as mannequins (placeholder users), and group settings and group membership are not migrated. Until mannequins are reclaimed, history is attributed to placeholders and content may not show in searches for the real user.

WHAT TO DO

Make sure every developer has a GitHub account in your enterprise (or managed user) before the waves. Map GitLab groups to GitHub teams in your Identity and licence mapping sheet, create the teams, and reclaim mannequins after each wave.

GitHub Docs: Reclaiming mannequins for GitHub Enterprise Importer

Budget real effort for CI, it does not migrate

COST LEVER

GitHub states that `.gitlab-ci.yml` has no automatic GitHub Actions equivalent in the importer, and CI/CD variables, job artifacts and pipeline triggers are not in the GitLab export. Actions Importer converts much of the YAML, but masked variables, artifact reports and cross-workflow caching need manual work.

WHAT TO DO

Run `gh actions-importer audit gitlab` before you set dates. Its report shows what converts fully, partially or not at all, so you can size the CI work per project. Keep GitLab runners until each project's Actions workflow is green.

TOOLS THAT CAN GET AROUND THIS

GitHub Actions Importer converts GitLab pipelines into GitHub Actions workflows and opens a pull request with them, with a list of manual steps such as creating secrets.

GitHub Actions Importer

Masked variable values and artifact reports still need manual work, and caching between jobs of different workflows is not supported.

Third-party tools, not endorsed by the platform vendor. Confirm in the tool's documentation and run a pilot first.

GitHub Docs: Migrating from GitLab with GitHub Actions Importer

Recreate protections and integrations by hand

COMPLIANCE

Repository policies (required approvals, merge trains, pipeline gates, mirroring), webhooks, snippets, issue boards and vulnerability reports are not migrated. A repository without branch protection on day one accepts direct pushes to main.

WHAT TO DO

Export a list of protected branches, approval rules, webhooks and integrations per project from GitLab. Recreate them as GitHub rulesets or branch protection, webhooks and apps straight after each wave, before developers switch.

GitHub Docs: Understand migrations from GitLab

Check organisation rulesets will not block the import

SAVES TIME

GitHub notes that organisation rulesets, such as commit author email rules, can make GEI migrations fail when old commits break them.

WHAT TO DO

Migrate into an organisation without blocking rulesets, or exclude the migrated repositories from them, then apply the rulesets after the migration.

GitHub Docs: Understand migrations from Azure DevOps to GitHub

FINISH SAFELY

Check it worked, or roll back

Full validation checklist

8 checks

Run these before you declare the migration complete. Record the results in the data validation report.

Check	How to check	What good looks like
Branches, tags and commits match	In a fresh clone from each side run `git branch -r | wc -l`, `git tag | wc -l` and `git rev-list --all --count`, and compare.	Counts are identical, and the default branch head commit hash is the same in GitLab and GitHub.
Issues and pull requests arrived	Compare open and closed issue counts and merge request counts in GitLab with issues and pull requests on GitHub, using `gh issue list --state all --limit 5000 | wc -l` and `gh pr list --state all --limit 5000 | wc -l` in the repository.	Counts match, and sampled pull requests keep their comments, reviewers and approvals.
Migration logs are clean	Read the migration log for each repository downloaded in Step 4.	No errors, and any warnings are for items you listed as not migrating.

Check	How to check	What good looks like
LFS content is real	Clone the repository from GitHub and run `git lfs ls-files`, then open a few of the listed files.	Files download with their content, not as text pointer files.
People and permissions are right	Check the organisation's mannequin list (People > Mannequins) and each repository's **Collaborators and teams** page.	No unclaimed mannequins for current staff, and each team has the role in your mapping sheet.
CI runs in Actions	Open the repository's **Actions** tab after merging the converted workflow, and trigger a run on a test branch.	The workflow runs green on push and on pull request, and deployments reach the same targets as in GitLab.
Protections and webhooks are in place	Try a direct push to the default branch from a developer account, and send a test delivery from each webhook.	The push is rejected by the ruleset, and each webhook shows a successful delivery.
GitLab is no longer written to	In GitLab, check the last activity date on each migrated project.	No pushes or merge requests after the freeze started.

Rollback plan

If it goes wrong —

ROLL BACK IF

- The production run fails for key repositories and cannot be rerun inside the freeze window
- Builds or deployments cannot run in GitHub Actions and the GitLab pipelines are still needed to ship
- History or pull requests are missing in a way the project owner will not accept

- Developers cannot authenticate to GitHub (SSO, tokens, managed users) on the switch day

POINT OF NO RETURN

GEI copies from GitLab and leaves the projects in place, so until developers push to GitHub you can unarchive GitLab and carry on. Once people commit, open pull requests and run releases on GitHub, that work exists only there and has to be pushed back by hand. Deleting GitLab projects, removing runners or cancelling GitLab cannot be undone, and GitLab-only data (pipeline history, job artifacts, snippets, issue boards) goes with it.

1

Unfreeze the GitLab projects

Where: GitLab > each project > Settings > General > Advanced > Unarchive project (or restore developer access)

Unarchive the wave's projects and tell developers to keep working in GitLab.

✓ **You should see:** Developers can push and open merge requests in GitLab again.

2

Push back any commits made on GitHub

Where: Terminal on your migration machine

For each repository where people pushed to GitHub, fetch from GitHub and push the new branches and tags to GitLab. Recreate any pull requests opened on GitHub as merge requests.

```
git clone --mirror https://github.com/contoso/payments-api.git
cd payments-api.git
git push --all https://gitlab.contoso.com/platform/payments/payments-api.git
git push --tags https://gitlab.contoso.com/platform/payments/payments-api.git
```

✓ **You should see:** GitLab shows every branch and tag that was created on GitHub.

3

Archive the GitHub copies

Where: GitHub > each repository > Settings > Danger Zone > Archive this repository

Archive the migrated repositories so nobody keeps using them, or delete them if you will rerun the migration (a rerun needs the target name free).

✓ **You should see:** The repositories show as archived and read-only.

4

Revert remotes and pipelines

Where: Each developer's clone, and your CI settings

Set each clone's origin back to the GitLab URL with `git remote set-url origin`, and re-enable any GitLab pipelines or runners you paused.

✓ **You should see:** git remote -v shows GitLab and pipelines run there again.

Agree the triggers and who decides before the go / no-go meeting. Use the rollback plan template.

Common problems and fixes

6 errors

`Migration fails: repository or archive too large`

Cause: The repository is over the 40 GiB GEI limit, a file is over 400 MiB, or the GitLab.com export archive is over 40 GB.

Fix: Clean history or move large files to LFS in GitLab, then migrate again.

`pack exceeds maximum allowed size`

Cause: A single push or commit is over GitHub's 2 GiB limit.

Fix: Split the history or remove the large commit before migrating.

`Export incomplete or users missing (self-managed)`

Cause: The GitLab token does not belong to an administrator, or project export is disabled.

Fix: Use an administrator's token with api and read_repository, enable project export, and migrate again.

`Token rejected by GitHub`

Cause: A fine-grained token was used, a scope is missing, or the organisation restricts classic tokens or needs SAML authorisation.

Fix: Use a classic token with the documented scopes, authorise it for SSO, and allow classic tokens for the migration.

`Migration fails on an organisation ruleset`

Cause: An organisation ruleset (for example commit author email rules) rejects old commits.

Fix: Exclude the migrated repositories from the ruleset during migration, then apply it afterwards.

`Build fails after migration: LFS files are pointers`

Cause: LFS objects are not migrated.

Fix: Run git lfs fetch --all from GitLab and git lfs push --all to GitHub.

Search the full error library →

FAQ

Frequently asked questions

Does GitHub Enterprise Importer support GitLab? −

Yes. GitHub made GitLab migrations with the gh gl2gh extension generally available in August 2026, from GitLab.com and currently maintained GitLab Self-Managed versions to GitHub Enterprise Cloud.

Are merge requests and issues migrated?

Yes. Issues migrate with comments, state and milestone events, and merge requests become pull requests with comments, reviewers, approvers and state events.

Can I migrate GitLab to GitHub Enterprise Server?

Not with GEI. GitHub says GitLab migrations to GitHub Enterprise Server are not supported; for code only, use a mirror push.

Does GitLab CI move to GitHub Actions?

Not with GEI. Use GitHub Actions Importer to convert .gitlab-ci.yml into workflows, then finish the parts it marks as partial or manual.

Can I migrate in stages and sync changes later?

No. The importer has no delta migration, so halt work in each project during its production run.

What if I only need the code?

Use GitHub Importer or git clone --mirror and git push --mirror. Both keep history but not issues or merge requests.

SOURCES

Official sources

11 official pages this guide is based on

Vendors change limits and features, so confirm details on the source before you rely on them. Last checked 4 Nov 2026.

- GitHub Docs: Migrating from GitLab to GitHub
- GitHub Docs: Understand migrations from GitLab

- GitHub Docs: Manage access for a GitLab migration
- GitHub Docs: Install and configure GitHub Enterprise Importer (GitLab)
- GitHub Docs: Migrate your repositories from GitLab
- GitHub Docs: Migrate from GitLab with the GraphQL API
- GitHub Changelog: Migrate from GitLab to GitHub with GitHub Enterprise Importer
- GitHub Docs: Reclaiming mannequins for GitHub Enterprise Importer
- GitHub Docs: Migrating from GitLab with GitHub Actions Importer
- GitHub Docs: Archiving repositories
- GitHub: GEI CLI extension releases (gl2gh, ado2gh)