The GitHub Guide to Code Migration

Migrations can be about more than just switching from one version control system to another; it's your chance to refresh, correct old inconsistencies and bottlenecks, and implement modern and collaborative workflows that will make developers' work easier.

In addition to rethinking workflows, a migration is an opportunity to bring together formerly separate teams, business units, and products under one easily-manageable and highly-collaborative umbrella. Not only does this ease administrative headaches, the added transparency around code development and review can make for smooth sailing when auditors come calling.

This guide reviews GitHub's suggested workflow structure, best practices for a successful migration, and an overview of how to get support if you need it.

Best Practices for Migrations

It's tempting to move every team over to GitHub as soon as you get started. But, to ensure your migration is painless there are some easy steps that we recommend completing first:

USER EDUCATION

Change is difficult, especially if you are not predisposed to it—and some developers have their workflows seemingly hard-coded in their minds. Whether you've already created a curriculum or FAQ page detailing the benefits of migrating to GitHub, or you want to take advantage of GitHub's complimentary training options, change is easier when everyone is on the same page...or branch, in this case.

COMMUNICATION WITH USERS IS PARAMOUNT

As mentioned above, change can be challenging to implement, and major changes that take people by surprise may be met with resistance. Moving to GitHub is all about making developer lives better—it's something to shout from the rooftops! Or at least announce in advance. Clear and consistent communications about the migration process (including explicit dates or windows) can give everyone plenty of time to adapt and prepare.

TEST, TEST, TEST

With extensive practice runs, there shouldn't be any surprises during the actual migration. While we can't make any guarantees, testing early and often, and more importantly, documenting both the user and administrative processes during tests, can reduce headaches later on.

DON'T MIGRATE EVERYTHING AT ONCE

Certain repositories and teams will be easier to migrate than others due to a host of factors like history, current workflow, and current projects. Don't try to force a migration teams aren't prepared for. One approach is to offer select teams or users who are already familiar with Git a migration "pilot" spot; we've seen this method used successfully as a way to highlight innovation areas within companies.

KEEP REPOSITORY BEST PRACTICE IN MIND

Freezing source repositories (which means preventing new changes from being committed to those repositories) with a clearly-communicated freeze date can help reduce the chance of information "falling through the cracks". Also, smaller repositories make migration easier—1 GB or less is ideal. You can take advantage of tools such as git-filter-branch and BFG Repo Cleaner to remove large files or track them with Git LFS.

START NEW PROJECTS ON GITHUB

While it can be difficult to force a team to migrate in the middle of a sprint, you can minimize the impact of the change by starting all new projects on GitHub instead. Not only will it present the opportunity for a clean break, but it will also help teams get familiar with GitHub Flow by working on a project from start to finish on GitHub.

DON'T FORGET YOUR INTEGRATIONS

A common mistake made during migrations is leaving off integrations. With so much to port over including repositories, teams, and more, they can be easy to forget—even if they are vital components such as Slack, JIRA, Jenkins, etc. Thorough auditing of elements to migrate can help prevent this.

SET SUCCESS INDICATORS AND MEASURE

A migration doesn't end when the last team joins your GitHub instance. Ensuring that each team and project is thriving on GitHub happens through continued measurements that show where the project is successful, and less so. While performance indicators vary, here are a few ideas to get you started:

- Percentage of migrated notified teams
- Satisfaction of teams with the migration process
- Usage metrics from GitHub
- Average migration time in days

DIY vs GitHub Assisted Migrations

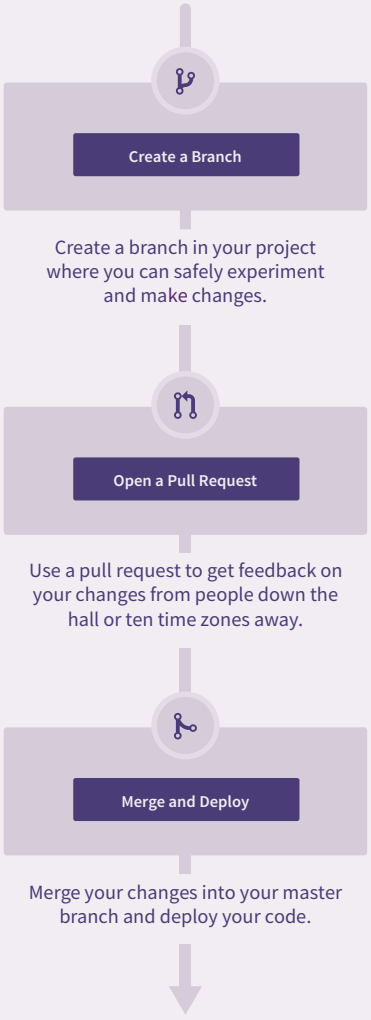
For many people, migration is simply a matter of following along with our extensive documentation. However, if you have several Organizations, multiple vital integrations, or perhaps just need a little assistance, GitHub's Professional Services team can help. The team provides support before, during, and after migration—not only with the migration itself, but also with understanding GitHub Flow, and helping to build highly collaborative and transparent workflows.

If you would like to learn more about GitHub's Professional Services, please reach out to your account manager and they can schedule a meeting. Additionally, they can share more information about our new Migration Package, which provides in-depth team training and migration assistance.

The GitHub Flow

To help developers get the most out of GitHub, they'll need to understand the benefits of the GitHub Flow and how it fits into your organization's development practices. A migration is a chance to adjust prevailing cultural attitudes around developing code—not an opportunity to spend resources rebuilding bad practices into Git.

The GitHub Flow makes developer workflows simpler, more collaborative, and more transparent. It's a lightweight, branch-based workflow that supports teams and projects where deployments are made regularly. While some Git knowledge is required, it's a process that works to simplify your development lifecycle by putting pull requests front and center.



GitHub

Questions about using GitHub? We're here to help.
Get in touch with:

sales@github.com
1.877.448.4820