# How to Rebase and Sync Your Fork with Upstream in Git

## Introduction

When contributing to open source projects like Mautic, it's standard to have a personal fork of the repo.

Other people do merge in their pull requests (PRs) over time, however, so your fork falls behind the main repository (or "upstream").

This article explains how to:

- Keep your fork current,
- Rebase your feature branches, and
- Fix synchronization issues before opening a PR.

Whether you are a newcomer to Git or have been working with it for many years, this guide will keep you on the best practices of open-source collaboration.

## What is “Upstream” and why it matters

When you fork Mautic’s repository, you’re essentially making your own copy of it under your GitHub account.

| Term | Meaning |
|---|---|
| Upstream | The original Mautic repository https://github.com/mautic/mautic |
| Origin | Your personal fork of that repo (e.g. https://github.com/YOUR-GITHUB-USERNAME/mautic) |
| Local | The copy on your computer that you work on. |

Keeping your local and origin branches up to date with upstream ensures that your PRs merge smoothly without conflicts

## Before You Begin

Make sure you have:

- Git installed

- A cloned fork of Mautic on your local machine
- A terminal or Git Bash ready

Verify your remotes:

```
git remote -v
```

Expected output (before setting upstream):

```
origin  https://github.com/yourusername/mautic.git (fetch)
origin  https://github.com/yourusername/mautic.git (push)
```

# Setting up your upstream remote

To connect your fork to Mautic's main repo, add an upstream remote:

```
git remote add upstream https://github.com/mautic/mautic.git
```

Confirm it's added correctly:

```
git remote -v
```

You should now see:

```
origin  https://github.com/yourusername/mautic.git (fetch)
origin  https://github.com/yourusername/mautic.git (push)
upstream  https://github.com/mautic/mautic.git (fetch)
upstream  https://github.com/mautic/mautic.git (push)
```

# Steps syncing your fork with Upstream

When the upstream repository changes, you'll want to update your local default branch and then push it to your fork.

Step 1: Fetch all branches from upstream

```
git fetch upstream
```

Step 2: Switch to your local default branch

```
git checkout origin/7.x
```

Step 3: Merge upstream changes

```
git merge upstream/x.x
```

or (recommended):

```
git rebase upstream/7.x
```

Step 4: Push changes to your fork

```
git push origin 7.x
```

Now your fork's 7.x branch is up to date with Mautic's.

# Rebasing your pull request (PR)

When working on a feature branch, such as feature/improve-login, it's important to rebase it on top of the latest state of the default branch.

The default branch is the branch in a repository where new changes are merged.
In many open-source projects, it's called main, but in Mautic it may have a versioned name such as 7.x or 5.2, depending on the development cycle.

Screenshot: The default branch indicator 7.x on Mautic's GitHub repository.

Now that you know how to identify the default branch, let's go through the steps to rebase your work properly.
This ensures your feature branch contains the most recent updates from Mautic before you push your changes or open a pull request.

Step 1: Fetch upstream updates

```
git fetch upstream
```

Step 2: Switch to your feature branch

```
git checkout feature/improve-login
```

Step 3: Rebase onto upstream/main

```
git rebase upstream/7.x
```

During rebasing, Git will pause if there are conflicts. Resolve them and continue with:

```
git add .
git rebase --continue
```

If you want to abort the process:

```
git rebase --abort
```

Step 4: Push the rebased branch to your fork

Since history changed, use the --force flag carefully:

```
git push origin feature/improve-login --force
```

# Resetting your repo to match upstream

If your local repo gets messy or too far behind, you can completely reset it to match the upstream repository.

Warning: This will overwrite local changes.

```
git fetch upstream
git checkout 7.x
git reset --hard upstream/7.x
git push origin 7.x --force
```

This command sequence aligns your local and remote main branches exactly with upstream  no extra commits, no drift.

# Dealing with Merge Conflicts

Sometimes when rebasing or merging, Git may report a conflict,  this means that the same part of a file was changed both in your branch and in the upstream branch. Git can't automatically decide which version to keep, so it asks you to make that decision manually.

Example output:

```
Auto-merging src/User/Login.php

CONFLICT (content): Merge conflict in src/User/Login.php
```

## How to Resolve

i. Open the conflicting file listed in the message.

ii. Look for conflict markers like these:

```
<<<<<<< HEAD

Your version of the code

=======

The version from upstream (or the default branch)

>>>>>>> upstream/<default-branch>
```

iii. Compare both sections carefully.

- The part between <<<<<<< HEAD and ======= is your local change.
- The part between ======= and >>>>>>> upstream/<default-branch> is the change from upstream.

iv. Decide which version to keep, or combine both if that makes the most sense.

- Keep your version if it still applies correctly to the new upstream code.
- Keep the upstream version if it contains improvements, bug fixes, or refactors that your branch should now follow.
- Sometimes, you may need to manually merge both versions into one clean and consistent block of code.

v. Remove the conflict markers (<<<<<<<, =======, >>>>>>>) after editing.

vi. Stage the resolved file and continue the rebase:

```
git add src/User/Login.php

git rebase --continue
```

Note: When you resolve merge conflicts, it’s a good practice to note what you changed in your pull request description or comments.
This helps reviewers understand what you decided and why.

For example, you might include something like this in your PR comment:

Merge conflict resolution notes:

- Resolved conflict in src/User/Login.php line 120–140.
- Kept upstream changes for improved logging.
- Reapplied my fix for user token validation after the refactor.

This level of transparency allows reviewers to check the affected lines more easily and confirm that the resolution aligns with project standards.

# Common Mistakes and Fixes

| Problem | Cause | Solution |
| --- | --- | --- |

| | | |
|---|---|---|
| fatal: No upstream configured | You haven’t added the upstream remote | Run git remote add upstream https://github.com/mautic/mautic.git |
| force push rejected | Branch is protected or you lack permission | Check PR branch permissions |
| Conflicts during rebase | Code differences between upstream and your branch | Resolve conflicts and continue |
| Rebase fails completely | Rebase was aborted | Try again or reset with git reset --hard upstream/main |

# Need help?

If you get stuck while rebasing or syncing your fork, reach out to the Mautic Product Team on Slack:

Channel: #t-product
You’ll find friendly maintainers who can help guide you through Git issues.

# Conclusion

Rebasing and syncing might appear intimidating at first but is in fact one of the most useful Git skills when contributing to open source projects such as Mautic. Having your fork synced to upstream simplifies the lives of maintainers and makes your contributions merge-ready at all times.