How to rebase branch and sync forked repository with upstream using Git

Introduction

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

Maintainers merge pull requests (PRs) over time, so your fork might fall behind the original repository.

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 keeps you up to date on the best practices of open source collaboration.

What is upstream, origin, and local repository

If you're not yet familiar with open source repository terms, take a look at the table below.

Term	**Meaning**	
Upstream	The original repository	

Origin	The forked repository	
Local	The copy of the remote (upstream or origin) repository on the local machine	

When you fork Mautic's repository, you're essentially making a copy of the upstream under your GitHub account. Then, you should always clone your forked repository on your local machine rather than the upstream repository, because most open source projects don't allow you to push changes directly to the upstream.

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

Prerequisites

Make sure you:

- Installed a code editor, such as VS Code
- Installed Git
- Cloned your forked repository on your local machine

The next step is to verify your remote repositories by running the following command:

```
git remote -v
```

Expected output (before setting up upstream):

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

```
ps://github.com/yourusername/mautic.git (push)
```

Setting up remote upstream

To connect your fork to the original repository, add upstream to your remote repository:

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

Then, run the command below to confirm if 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://g
ithub.com/mautic/mautic.git (fetch)upstream  https://github.com/mauti
c/mautic.git (push)
```

Syncing the forked repository with the upstream

When the upstream repository has new changes, you need to update your local default branch and push it to your forked repository.

Note: for the tutorials across in this article, it's assumed that the default branch name is 7.x. If the default branch name differs, change it to the correct one.

Step 1: Fetch all branches from upstream

```
git fetch upstream
```

Step 2: Switch to the local default branch

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

Step 3: Rebase the local default branch with upstream

Rebasing your branch with upstream means merging the new changes from upstream into your local branch, creating a new commit for each commit in the original branch:

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

Step 4: Push changes to the forked repository

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

Now your local and forked repository's 7.x branch is up to date with the upstream.

Rebasing PR

When working on a feature branch, you need to rebase it on top of the latest state of the default branch.

The default branch in a repository is where new changes are merged. In many open source projects, the default branch is typically called main. However, Mautic uses versioned names such as 6.0, 7.x, and so on, depending on the current development cycle.

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

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

Step 1: Fetch branches from upstream

```
git fetch upstream
```

Step 2: Switch to the feature branch

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

Change the feature/improve-login to your branch name.

Step 3: Rebase the feature branch with upstream

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

During rebasing, Git pauses when merge conflicts occur, and you must resolve them before continuing.

If you're not familiar with resolving merge conflicts, please read the "Dealing with merge conflicts" section. Once all conflicts are resolved, you can continue to rebase by running these commands:

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

If you want to abort the rebasing process, run the following command:

```
git rebase --abort
```

Step 4: Push the rebased branch to the forked repository

Since rebasing changes the history, you want to use the --force flag carefully as below:

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

Resetting local and forked repositories to match the upstream

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

Warning: Running commands below overwrites local changes. So, please do it with caution.

```
git fetch upstreamgit checkout 7.xgit reset --hard upstream/7.xgit pu
sh origin 7.x --force
```

This command sequence perfectly syncs your local and forked default branches with upstream, ensuring no extra commits or drift.

Rebase: common mistakes and fixes

Problem	Cause	Solution
fatal: No upstream configured	You haven't added the upstream remote	Run: `git remote add upstrea m https://github.com/m autic/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

Introduction

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

Maintainers merge pull requests (PRs) over time, so your fork might fall behind the original repository.

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 keeps you up to date on the best practices of open source collaboration.

What is upstream, origin, and local repository

If you're not yet familiar with open source repository terms, take a look at the table below.

Term	Meaning	
Upstream	The original repository	

Origin	The forked repository	
Local	The copy of the remote (upstream or origin) repository on the local machine	

When you fork Mautic's repository, you're essentially making a copy of the upstream under your GitHub account. Then, you should always clone your forked repository on your local machine rather than the upstream repository, because most open source projects don't allow you to push changes directly to the upstream.

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

Prerequisites

Make sure you:

- Installed a code editor, such as VS Code
- Installed Git
- Cloned your forked repository on your local machine

The next step is to verify your remote repositories by running the following command:

```
git remote -v
```

Expected output (before setting up upstream):

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

```
ps://github.com/yourusername/mautic.git (push)
```

Setting up remote upstream

To connect your fork to the original repository, add upstream to your remote repository:

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

Then, run the command below to confirm if 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://g
ithub.com/mautic/mautic.git (fetch)upstream  https://github.com/mauti
c/mautic.git (push)
```

Syncing the forked repository with the upstream

When the upstream repository has new changes, you need to update your local default branch and push it to your forked repository.

Note: for the tutorials across in this article, it's assumed that the default branch name is 7.x. If the default branch name differs, change it to the correct one.

Step 1: Fetch all branches from upstream

```
git fetch upstream
```

Step 2: Switch to the local default branch

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

Step 3: Rebase the local default branch with upstream

Rebasing your branch with upstream means merging the new changes from upstream into your local branch, creating a new commit for each commit in the original branch:

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

Step 4: Push changes to the forked repository

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

Now your local and forked repository's 7.x branch is up to date with the upstream.

Rebasing PR

When working on a feature branch, you need to rebase it on top of the latest state of the default branch.

The default branch in a repository is where new changes are merged. In many open source projects, the default branch is typically called main. However, Mautic uses versioned names such as 6.0, 7.x, and so on, depending on the current development cycle.

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

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

Step 1: Fetch branches from upstream

```
git fetch upstream
```

Step 2: Switch to the feature branch

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

Change the feature/improve-login to your branch name.

Step 3: Rebase the feature branch with upstream

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

During rebasing, Git pauses when merge conflicts occur, and you must resolve them before continuing.

If you're not familiar with resolving merge conflicts, please read the "Dealing with merge conflicts" section. Once all conflicts are resolved, you can continue to rebase by running these commands:

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

If you want to abort the rebasing process, run the following command:

```
git rebase --abort
```

Step 4: Push the rebased branch to the forked repository

Since rebasing changes the history, you want to use the --force flag carefully as below:

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

Resetting local and forked repositories to match the upstream

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

Warning: Running commands below overwrites local changes. So, please do it with caution.

```
git fetch upstreamgit checkout 7.xgit reset --hard upstream/7.xgit pu
sh origin 7.x --force
```

This command sequence perfectly syncs your local and forked default branches with upstream, ensuring no extra commits or drift.

Rebase: common mistakes and fixes

Problem	Cause	Solution
fatal: No upstream configured	You haven't added the upstream remote	Run: `git remote add upstrea m https://github.com/m autic/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 upstr eam/main`

Dealing with merge conflicts

Sometimes, when rebasing or merging, Git may report a conflict. This means the same part of

a file was changed in both your branch and 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.phpCONFLICT (content): Merge conflict in src/User/Login.php
```

How to resolve merge conflicts

1. Open the conflicting file listed in the message.

2. Look for conflict markers like these:

```
<<<<<<< HEADYour version of the code=======The version from upstream (or the default branch)>>>>>>> upstream/<default-branch>
```

3. 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.

4. 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 the two versions into a single, clean, consistent block of code.

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

6. Stage the resolved file and continue the rebase:

```
git add src/User/Login.phpgit rebase --continue
```

Note: when resolving merge conflicts, it’s a good practice to document what you changed in your PR 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 more easily check the affected lines and confirm that the resolution aligns with project standards.

Need help?

If you get stuck while rebasing or syncing your fork, you can reach out to the Mautic Product Team on Slack at the #t-product channel.

Conclusion

Rebasing and syncing might appear intimidating at first, but they are, in fact, among the most useful Git skills when contributing to open source projects such as Mautic. Having your fork synced with upstream simplifies maintainers' lives and keeps your contributions merge-ready at all times.

Online URL: https://kb.mautic.org/article/how-to-rebase-branch-and-sync-forked-repository-with-upstream-using-git.html

SyntaxHighlighter.config.stripBrs=true; SyntaxHighlighter.all();