Git commands to run to fix commit history.

how do I fix my git history in a pr where there is merged commits so I can’t squash

If you're in a situation where you can't squash commits in a pull request (PR) due to merged commits or other complexities in your Git history, you can still clean up your Git history by performing a "rebase interactive" and then force pushing the changes. Here's a step-by-step guide to help you fix your Git history in this scenario:

1. **Create a Backup Branch (Optional)**: Before making any changes, consider creating a backup branch that points to the current state of your PR branch. This is a safety net in case something goes wrong.

   ```
   git checkout <pr-branch>
   git checkout -b backup-pr-branch
   ```

2. **Rebase Interactive**: Use an interactive rebase to modify your commits. Assuming your PR branch is named `<pr-branch>`:

   ```
   git checkout <pr-branch>
   git rebase -i HEAD~<number-of-commits>
   ```

   Replace `<number-of-commits>` with the number of commits you want to edit. This will open an interactive rebase window.

3. **Edit and Reorder Commits**: In the interactive rebase window, you'll see a list of commits along with commands you can perform. For each commit you want to modify, change `pick` to `edit`. Save and close the rebase window. Note: Press `i` to start editing. Then `ctrl + o` to stop editing and type `:wq` to save

4. **Edit Commits**: Git will now go through each commit marked as "edit." For each commit, Git will stop and allow you to make changes.

   - To squash commits, modify the commit message to include "fixup!" or "squash!" before the commit message. Git will automatically combine these commits during the rebase.

   - To edit the commit, make your changes, then use `git commit --amend` to finalize your changes. You can also use `git add` to stage additional changes before amending.

   - To reorder commits, you can manually change the order in the rebase window or move commits around.

5. **Continue Rebase**: After making your changes to each commit, continue the rebase by running:

   ```
   git rebase --continue
   ```

   If conflicts arise during the rebase, resolve them as usual with `git add` and `git rebase --continue`.

6. **Force Push**: Once you've finished editing and squashing commits, you'll need to force push the changes to your PR branch. This overwrites the remote history with your new commits.

   ```
   git push origin <pr-branch> --force
   ```

   Be cautious with force pushes, as they rewrite history and can affect collaborators who have based their work on the existing commits.

Remember that this process rewrites history, and it's important to coordinate with your team if others are collaborating on the same branch. If you're working in a shared repository, it's good practice to communicate your intentions and coordinate any changes that affect the shared branch.

what if I pushed the code

If you've already pushed the code with unwanted or incorrect commits, and you want to fix the Git history on the remote repository, you'll need to be careful because rewriting history can cause issues for collaborators who have already based their work on the existing history. Here's a general approach you can consider:

1. **Notify Collaborators**: Before making any history-changing actions, communicate with your collaborators about your intentions. Let them know that you're planning to rewrite history and force push to the remote repository. This way, they can adjust their local repositories accordingly.

2. **Create a Backup Branch**: If you're unsure about the changes you'll make, create a backup branch from the current state of the remote branch:

   ```
   git checkout <branch-name>
   git checkout -b backup-branch
   ```

3. **Perform Interactive Rebase Locally**: On your local machine, perform an interactive rebase to clean up the commits as you want them:

   ```
   git checkout <branch-name>
   git rebase -i HEAD~<number-of-commits>
   ```

   Follow the steps for editing, squashing, or modifying commits as needed.

4. **Force Push to Remote**: After cleaning up your local history, force push your changes to the remote repository. This will overwrite the remote branch's history with your new history:

   ```
   git push origin <branch-name> --force
   ```

   Important: Force pushing rewrites history, which can cause problems for others collaborating on the same branch. Use this approach with caution and communicate with your team.

5. **Notify Collaborators Again**: After the force push, communicate with your collaborators once more, informing them that you've rewritten the history. Advise them to rebase or merge their local branches to the updated remote branch.

6. **Cleanup and Maintain History Discipline**: In the future, consider adopting a disciplined approach to maintain a clean history by squashing or amending commits before pushing them. This can help avoid the need for massive history rewrites.

Remember that force pushing can disrupt the work of other contributors who have based their work on the original history. It's important to follow good communication practices and coordinate with your team to minimize disruptions and ensure a smooth transition.