Using Git on GitHub Docs

In this article

You can use Git on the command line to commit changes and then push them to the documentation repository.

> Articles in the "Contributing to GitHub Docs" section refer to the documentation itself and are a resource for GitHub staff and open source contributors.

This article describes the process of creating a topic branch for the documentation repository, committing changes, and pushing your changes back up to the remote repository.

The article assumes you have already cloned the documentation repository locally and you will be making changes on your local computer rather than on GitHub.com or in a codespace. For more information, see "Cloning a repository."

Setting up your topic branch and making changes

To keep your local branches in sync with their remotes and avoid merge conflicts, follow these steps as you work on documentation.

1. In the terminal, change the current working directory to the location where you cloned the documentation repository. For example:

   ```
   cd ~/my-cloned-repos/docs
   ```

2. Switch to the default branch: `main`.

   ```
   git checkout main
   ```

3. Get the most recent commits from the remote repository.

   ```
   git pull origin main
   ```

4. Switch to or create a topic branch.

   - To start a new project, create a new topic branch from `main`.

     ```
     git checkout -b YOUR-TOPIC-BRANCH
     ```

> **Note**: You can use forward slashes as part of the branch name, for example to include your user name:
>
> ```
> git checkout -b my-username/new-codespace-policy
> ```

- To work on an existing project, switch to your topic branch and merge changes from `main`.

  ```
  git checkout YOUR-TOPIC-BRANCH
  git merge main
  ```

  If you run into merge conflicts, follow the steps later in this article for resolving merge conflicts.

5. Open your preferred text editor, edit files as required, then save your changes.

Committing and pushing your changes

1. When you're ready to commit your changes, open a terminal and check the status of your topic branch with `git status`. Make sure you see the correct set of changes.

   ```
   git status
   On branch YOUR-TOPIC-BRANCH

   Changes not staged for commit:
     (use "git add <file>..." to update what will be committed)
     (use "git checkout -- <file>..." to discard changes in working directory)
           deleted:    example-deleted-file.md
           modified:   example-changed-file.md

   Untracked files:
     (use "git add <file>..." to include in what will be committed)
           example-new-file.md
   ```

2. Stage the changed files so that they're ready to be committed to your topic branch.

   - If you created new files or updated existing files, use `git add FILENAME [FILENAME...]`. For example:

     ```
     git add example-new-file.md example-changed-file.md
     ```

     This adds the updated version of the files to Git's staging area, from which changes can be committed. To unstage a file, use `git reset HEAD FILENAME`. For example, `git reset HEAD example-changed-file.md`.
   - If you deleted files, use `git rm FILENAME [FILENAME...]`. For example:

     ```
     git rm example-deleted-file.md
     ```

3. Commit your changes.

   ```
   git commit -m "Commit message title (max 72 characters)

   Optional fuller description of what changed (no character limit).
   Note the empty line between the title and the description,
   and the closing quotation mark at the end of the commit message."
   ```

This commits the staged changes locally. You can now push this commit, and any other unpushed commits, to the remote repository.

To remove this commit, use `git reset --soft HEAD~1`. After running this command our changes are no longer committed but the changed files remain in the staging area. You can make further changes and then `add` and `commit` again.

4. Push your changes to the remote repository on GitHub.com.
    - The first time you push your branch you can choose to add an upstream tracking branch. This allows you to use `git pull` and `git push` on that branch without additional arguments.

      ```
      git push --set-upstream origin YOUR-TOPIC-BRANCH
      ```

    - If you've pushed this branch before, and set an upstream tracking branch you can use:

      ```
      git push
      ```

Best practices for commits

- Favor commits that contain small, focused groups of changes over commits with large, unfocused groups of changes, since this will help you write commit messages that other people can easily understand. An exception is the initial commit for a new project or category. These commits are sometimes large, as they often introduce the bare versions of many articles at once to provide an organizational scheme for subsequent work.
- If you are incorporating feedback or want to address a set of changes to a particular person or team for review, @mention the person whose suggestions you are incorporating. For example: "Incorporating feedback from @octocat," or "Updating billing configuration steps - cc @monalisa for accuracy."
- If a commit addresses an issue, you can reference the issue number in the commit, and a link to the commit will appear in the issue conversation timeline: "Addresses #1234 - adds steps for backing up the VM before upgrading."

  > **Note**: We generally don't close an issue via a commit. To close an issue, open a pull request and add "Closes #1234" to the description. The linked issue will be closed when the pull request is merged. For more information, see "Linking a pull request to an issue."

- Make commit messages clear, detailed, and imperative. For example: "Adds a conceptual article about 2FA," not "Add info."
- Try not to leave uncommitted changes in your local branch when you finish working for the day. Get to a good stopping point and commit and push your changes so your work is backed up to the remote repository.
- Only push up to GitHub.com after you've made a few commits. Pushing after every commit adds noise to our ops channels on Slack and causes unnecessary builds to run.

Resolving merge conflicts

When you try to merge two branches that contain different changes to the same part of a file, you will get a merge conflict. In our workflow, this most often occurs when merging

`main` down into a local topic branch.

There are two ways to handle merge conflicts:

- Edit the file in your text editor and choose which changes to keep. Then commit the updated file to your topic branch from the command line.
- Resolve the merge conflicts on GitHub.com.

Resolving merge conflicts by editing the file and committing the changes

1. On the command line, note the files that contains merge conflicts.
2. Open the first of these files in your text editor.
3. In the file, look for the merge conflict markers.

```
<<<<<<< HEAD
Here are the changes you've made.
=====================
Here are the changes from the main branch.
>>>>>>> main
```

4. Decide which changes to keep and delete the unwanted changes and the merge conflict markers. If you need to make further changes, you can do so at the same time. For example, you could change the five lines shown in the previous code sample to the single line:

```
Here are the changes you want to use.
```

   If there are multiple files with merge conflicts, repeat the previous steps until you resolve all conflicts.

   Note: You should apply care when resolving merge conflicts. Sometimes you will simply accept your own changes, sometimes you will use the upstream changes from the `main` branch, and sometimes you will combine both sets of changes. If you're unsure of the best resolution, be wary of replacing the changes from upstream as these may have been made for specific reasons that you're not aware of.

5. In the terminal, stage the file, or files, that you just modified.

```
git add changed-file-1.md changed-file-2.md
```

6. Commit the files.

```
git commit -m "Resolves merge conflicts"
```

7. Push the committed changes to the remote repository on GitHub.com.

```
git push
```

Creating a pull request

We recommend you open your pull request on GitHub early. Create the pull request as a draft until you are ready for it to be reviewed. Each time you push changes, your commits will be added to the pull request.

> **Note**: You can quickly access pull requests you've created by clicking **Pull requests** at the top of every page on GitHub.com.

For more information, see "Creating a pull request."