Documentation contributors guide

The topics in this section describe how you can contribute to this documentation. For information about contributing code to the Angular framework, see Contributing to Angular.

Angular is an open source project that appreciates its community support, especially when it comes to the documentation.

You can update the Angular documentation in these ways:

- Make a minor change
- Make a major change

IMPORTANT:

To submit changes to the Angular documentation, you must have:

- A GitHub account
- A signed Contributor License Agreement

Make a minor change

You can make minor changes to a documentation topic without downloading any software. Many common documentation maintenance tasks require only minor changes to a few words or characters in a topic. Examples of minor changes include:

- Correcting a typo or two
- Reviewing a topic and updating its review date
- Adding or updating search keywords

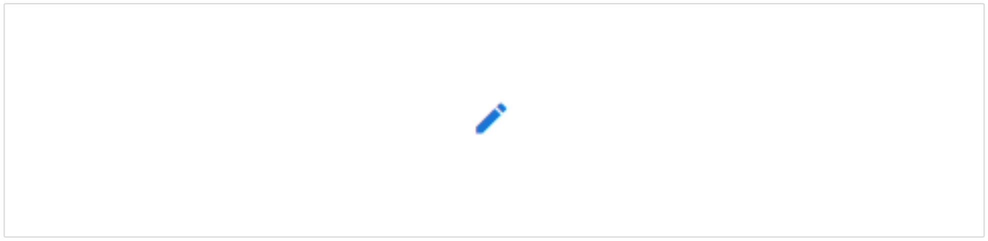
For more about keeping the documentation up to date, see Common documentation maintenance tasks.

To make larger changes to the documentation, you must install an Angular development environment on your local computer. You need this environment to edit and test your changes before you submit them. For information about configuring your local computer to make larger documentation updates, see Preparing to edit the documentation.

To make a minor change to a documentation topic

Perform these steps in a browser.

1. Confirm you have a signed Contributor License Agreement (CLA) on file. If you don't, sign a CLA.
2. Sign into github.com, or if you don't have a GitHub account, create a new GitHub account.
3. Navigate to the page in angular.io that you want to update.
4. On the page that you want to update, locate this pencil icon to the right of the topic's title



5. Click this icon to open the suggestion page.
6. In the suggestion page, in **Edit file**, update the content to fix the problem. If the fix requires more than correcting a few characters, it might be better to treat this as a major change.
7. Click the **Preview** tab to see how your markdown changes look when rendered. This view shows how the markdown renders. It won't look exactly like the documentation page because it doesn't display the text with the styles used in the documentation.
8. After you finish making your changes:

a. In **Propose changes**, enter a brief description of your changes that starts with `docs:` and is 100 characters or less in length. If necessary, you can add more information about the change in the larger edit window below the brief description.

b. Select **Create a new branch for this commit and start a pull request** and accept the default branch name.

c. Click **Propose changes** to open a pull request with your updated text.

After you open a pull request, the Angular team reviews your change and merges it into the documentation. You can follow the progress of your pull request in the pull request's page. You might receive a notification from GitHub if the Angular team has any questions about your change.

Make a major change

Making major changes or adding new topics to the documentation follows a different workflow. Major changes to a topic require that you build and test your changes before you send them to the Angular team.

These topics provide information about how to set up your local computer to edit, build, and test Angular documentation to make major changes to it.

- Overview of the Angular documentation editorial workflow
  Describes how to configure your local computer to build, edit, and test Angular documentation
- Documentation style guide
  Describes the standards used in the Angular documentation

Localize Angular documentation in a new language

Localizing Angular documentation is another way to contribute to Angular documentation. For information about localizing the Angular documentation in a new language, see Angular localization guidelines.

Last reviewed on Sun Dec 11 2022

Overview of documentation maintenance tasks

The Angular documentation needs routine maintenance to keep it up-to-date. The topics in this section describe routine maintenance tasks that you can perform to help keep the Angular documentation in good condition.

Documentation maintenance tasks fall into these two categories:

- Minor changes
- Major changes

Minor changes can be made in the GitHub site without the need to load any software or tools on your system. For information about making a minor change to the documentation, see Make a minor change.

Major changes require that you build and test your changes to the documentation on your local computer before you send them to the Angular documentation. For information about preparing your system to make major changes to the documentation, see Make a major change.

Routine documentation maintenance tasks

TASK	SCOPE
Review current documentation	Minor (See note)
Update search keywords	Minor
Resolve linter errors	Major
Resolve documentation issues	Major

NOTE:
Reviewing current documentation requires a minor change if all you need to do is update the `@reviewed` date. If you find a minor problem with a documentation topic, such as a typo, fixing it during your review is also a minor change.

If you find an issue that you don't feel comfortable fixing, open a docs issue in GitHub so someone else can fix it.

Last reviewed on Wed Oct 12 2022

Review documentation

You can review the Angular documentation, even if you have never contributed to Angular before. Reviewing the Angular documentation provides a valuable contribution to the community.

Finding and reporting issues in the documentation helps the community know that the content is up to date. Even if you don't find any problems, seeing that a document has been reviewed recently, gives readers confidence in the content.

This topic describes how you can review and update the Angular documentation to help keep it up to date.

To review a topic in angular.io

Perform these steps in a browser.

1. Find a topic to review by:
    a. Finding a topic with a **Last reviewed** date that is six months or more in the past.
    b. Finding a topic that has no **Last reviewed** date.
    c. Finding a topic that you've read recently.
2. Review the topic for errors or inaccuracies.
3. Complete the review.

a. If the topic looks good:

   i. Update or add the `@reviewed` entry at the end of the topic's source code.

   ii. Make a minor change to a documentation topic to publish the new reviewed date.

b. If you find an error that you don't feel comfortable fixing:

   i. Open a docs issue in GitHub.

   ii. Update or add the `@reviewed` entry at the end of the topic's source code.

   iii. Make a minor change to a documentation topic to publish the new reviewed date.

c. If you find an error that needs only a minor change:

   i. Update or add the `@reviewed` entry at the end of the topic's source code.

   ii. Make a minor change to a documentation topic to fix the error and save the new reviewed date.

d. If you find an error that needs major changes:

   i. Address the error:

      i. Make a major change, if you're comfortable, or

      ii. Open a docs issue in GitHub.

   ii. Whether you fix the error or open a new issue, update or add the `@reviewed` entry at the end of the topic's source code.

   iii. Make a minor change to a documentation topic to save the new reviewed date.

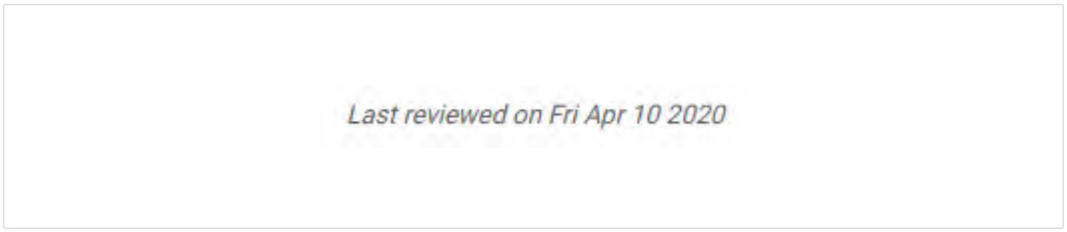
Find topics to review

You can review any topic in the Angular documentation, but these are the topics that benefit most from your review.

Topics that have not been reviewed in over six months

At the bottom of some topics, there's a date that shows when the topic was last reviewed. If that date is over six months ago, the topic is ready for a review.

This is an example of a **Last reviewed** date from the bottom of a topic. You can also see an example of this at the end of this topic.


Last reviewed on Fri Apr 10 2020

Topics that have never been reviewed

If a topic doesn't have a **Last reviewed** date at the bottom, it has never been reviewed. You can review such a topic and add a new **Last reviewed** date after you review it.

Topics that you know have a problem

If you know of a topic that has an error or inaccuracy, you can review it and make corrections during your review. If you don't feel comfortable fixing an error during your review, open a docs issue in GitHub. Be sure to add or update the **Last reviewed** date after you review the topic. Whether you fix the error or just open an issue, you still reviewed the topic.

Update the last reviewed date

After you review a topic, whether you change it or not, update the topic's **Last reviewed** date. The **Last reviewed** text at the bottom of the topic is created by the `@reviewed` tag followed by the date you reviewed the topic.

This is an example of an `@reviewed` tag at the end of the topic's source code as it appears in a code editor.

```
@reviewed 2022-09-08
```

The date is formatted as `YYYY-MM-DD` where:

- `YYYY` is the current year
- `MM` is the two-digit number of the current month with a leading zero if the month is 01 (January) through 09 (September)
- `DD` is the two-digit number of the current day of the month with a leading zero if the day is 01-09.

For example:

REVIEW DATE	`@REVIEWED` TAG	RESULTING TEXT DISPLAYED IN THE DOCS
January 12, 2023	`@reviewed 2023-01-12`	*Last reviewed on Thu Jan 12, 2023*
November 3, 2022	`@reviewed 2022-11-03`	*Last reviewed on Fri Nov 03, 2022*

Reviewing and updating a topic

These are the actions you can take after you review a topic.

The topic is accurate and has no errors

If the topic is accurate and has no errors, make a minor change to update the **Last reviewed** date at the bottom of the page. You can use the GitHub user interface to edit the topic's source code.

The topic requires minor changes

If the topic has minor errors, you can fix them when you make a minor change. Remember to update the **Last reviewed** date at the bottom of the page when you fix the error. For a minor change, you can use the GitHub user interface in a browser to edit the topic's source code.

The topic requires major changes

If the topic requires major changes, you can make a major change, or open a docs issue in GitHub. You shouldn't make major changes in the GitHub user interface because it doesn't allow you to test them before you submit them.

Whether you make the changes the topic needs or open a docs issue, you should still update the **Last reviewed** date. You can use the GitHub user interface in the browser if you only want to update the **Last reviewed** date.

Last reviewed on Sun Dec 11 2022

Update search keywords

You can help readers find the topics in the Angular documentation by adding keywords to a topic. Keywords help readers find topics by relating alternate terms and related concepts to a topic.

In angular.io, readers search for content by using:

- External search, such as by using google.com
- The search box at the top of each page

Each of these methods can be made more effective by adding relevant keywords to the topics.

To update search keywords in a topic

Perform these steps in a browser.

1. Navigate to the topic to which you want to add or update search keywords.
2. Decide what search keywords you'd like to add to the topic. Keywords should be words that relate to the topic and are not found in the topic headings.
3. Open the topic's **Edit file** page to make a minor change.
4. Add or update the `@searchKeywords` tag at the end of the topic with your keywords. The `@searchKeywords` tag takes a set of single-word keywords that are separated by spaces. The tag and the keywords must be enclosed in curly brackets. A sample tag is shown here to add these keywords to a page: *route*, *router*, *routing*, and *navigation*.

   ```
   {@searchKeywords route router routing navigation}
   ```

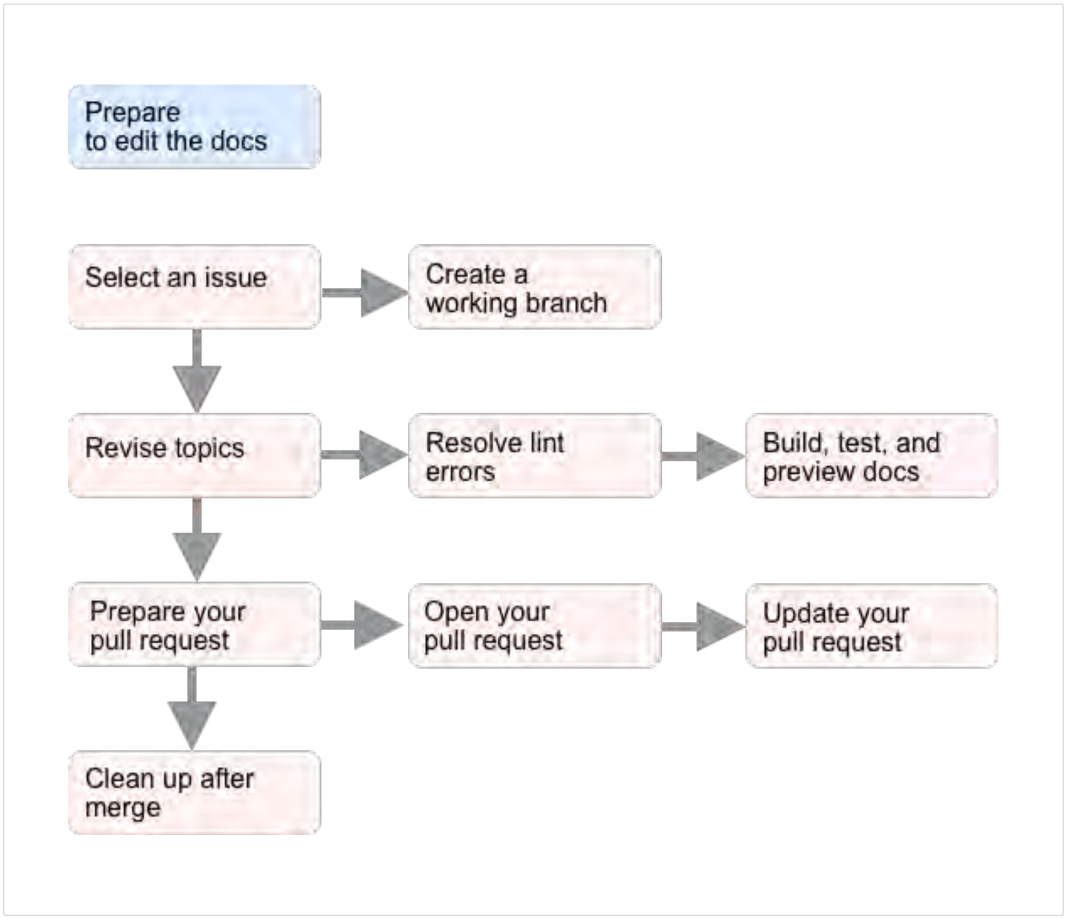
5. Update or add the `@reviewed` entry at the end of the topic's source code.
6. Propose your changes as described in make a minor change.

Last reviewed on Sun Dec 11 2022

Overview of Angular documentation editorial workflow

This section describes the process of making major changes to the Angular documentation. It also describes how Angular documentation is stored, built, revised, and tested.

The following diagram illustrates the workflow for revising Angular documentation. The steps are summarized below and described in the topics of this section.



Prepare to edit the docs

You perform this step one time to prepare your local computer to update the Angular documentation.

For more information about how to prepare to edit the docs, see Preparing to edit documentation.

Select a documentation issue

The first step in resolving a documentation issue is to select one to fix. The issue that you fix can be one from the list of documentation issues in the `angular/angular` repo or one you create.

For more information about how to select an issue to fix, see Selecting a documentation issue.

Create a documentation issue

If you want to fix a problem that has not already been described in an issue, open a documentation issue before you start. When you can relate an issue to your pull request, reviewers can understand the problem better when they review your pull request.

Create a working branch

After you select an issue to resolve, create a `working` branch in the `working` directory on your local computer. You need to make your changes in this branch to save and test them while you edit. After you fix the issue, you use this branch when you open the pull request for your solution to be merged into `angular/angular`.

For more information about how to create a `working` branch, see Starting to edit a documentation topic.

Revise topics

In your `working` branch, you edit and create the documentation topics necessary to resolve the issue. You perform most of this work in your integrated development environment (IDE).

For more information about how to revise a documentation topic, see Revising a documentation topic.

Resolve lint errors

Each time you save your edits to a documentation topic, the documentation linter reviews your topic. It reports the problems it finds in your topic to your IDE. To prevent delays later in the pull request process, you should correct these problems as they are reported. The documentation linter errors must be corrected before you open the pull request to pass the pull request review. Having lint errors in a topic can prevent the pull request from being approved for merging.

For more information about how to resolve lint problems in a documentation topic, see Resolving documentation linter messages.

Test your changes

As you edit documentation topics to resolve the issue you selected, you want to build a local version of the updated documentation. This is the easiest way to review your changes in the same context as the documentation's users.

You can also run some of the automated tests on your local computer to catch other errors. Running these tests on your local computer before you open a pull request speeds up the pull-request approval process.

For more information about how to build and test your changes before you open a pull request, see Building and testing documentation.

Prepare your pull request

To make your documentation changes ready to be added to the `angular/angular` repo, there are a few things to do before you open a pull request. For example, to make your pull request easy to review and approve, the commits and commit messages in your `working` branch must be formatted correctly.

For information about how to prepare your branch for a pull request, see Preparing documentation for a pull request.

Open your pull request

Opening a documentation pull request sends your changes to the Angular reviewers who are familiar with the topic. To be processed correctly, pull requests for `angular/angular` must be formatted correctly and contain specific information.

For information about how to format a pull request for your documentation update, see Opening a documentation pull request.

Update your pull request

You might get feedback about your pull request that requires you to revise the topic. Because the pull-request process is designed for all Angular code, as well as the documentation, this process might seem intimidating the first time.

For information about how to update your topics and respond to feedback on your changes, see Updating a documentation pull request in progress.

Clean up after merge

After your pull request is approved and merged into `angular/angular`, it becomes part of the official Angular documentation. At that point, your changes are now in the `main` branch of `angular/angular`. This means that you can safely delete your `working` branch.

It is generally a good practice to delete `working` branches after their changes are merged into the `main` branch of `angular/angular`. This prevents your personal fork from collecting lots of branches that could be confusing in the future.

For information about how to clean up safely after your pull request is merged, see Finishing up a documentation pull request.

Last reviewed on Wed Oct 12 2022

Prepare to edit Angular documentation

This topic describes the steps that prepare your local computer to edit and submit Angular documentation.

IMPORTANT:

To submit changes to the Angular documentation, you must have:

- A GitHub account
- A signed Contributor License Agreement

Complete a contributor's license agreement

Review Contributing to Angular. These sections are particularly important for documentation contributions:

1. Read the Angular Code of conduct
2. Read the Submission guidelines.

> **NOTE**:
> The topics in this section explain these guidelines specifically for documentation contributions.

3. Read and complete the Contributor license agreement that applies to you.

Install the required software

To edit, build, and test Angular documentation on your local computer, you need the following software. The instructions in this section assume that you are using the software in this list to complete the tasks.

Some software in this list, such as the integrated development environment (IDE), can be substituted with similar software. If you use a substitute IDE, you might need to adapt the instructions in this section to your IDE.

For more information about the required software, see Setting up the local environment and workspace.

- **Version control software**
  - Git command line
  - GitHub desktop (optional)
- **Integrated development environment**
  - Visual Studio Code
- **Utility software**
  - node.js

    Angular requires an active long-term-support (LTS) or maintenance LTS version of Node.js.
  - nvm
  - Yarn
  - Homebrew for macOS or Chocolatey for Windows
  - Vale (see note)

IMPORTANT:

Wait until after you clone your fork of the `https://github.com/angular/angular` repo to your local computer before you configure Vale settings.

You can also install other tools and IDE extensions that you find helpful.

Set up your workspaces

The Angular documentation is stored with the Angular framework code in a GitHub source code repository, also called a *repo*, at: https://github.com/angular/angular. To contribute documentation to Angular, you need:

- A GitHub account
- A *fork* of the Angular repo in your personal GitHub account.
  This guide refers to your personal GitHub account as `personal`. You must replace `personal` in a GitHub reference with your GitHub username. The URL: `https://github.com/personal` is not a valid GitHub account. For convenience, this documentation uses these shorthand references:
  - `angular/angular`
    Refers to the Angular repo. This is also known as the *upstream* repo.
  - `personal/angular`
    Refers to your personal fork of the Angular repo. Replace `personal` with your GitHub username to identify your specific repo. This is also known as the *origin* repo.
- A *clone* of your `personal/angular` repo on your local computer

GitHub repos are cloned into a `git` workspace on your local computer. With this workspace and required tools, you can build, edit, and review the documentation from your local computer.

When you can build the documentation from a workspace on your local computer, you are ready to make major changes to the Angular documentation.

For more detailed information about how to set up your workspace, see Create your repo and workspaces for Angular documentation.

For more detailed information about how to build and test the documentation from your local computer, see Build and test the Angular documentation.

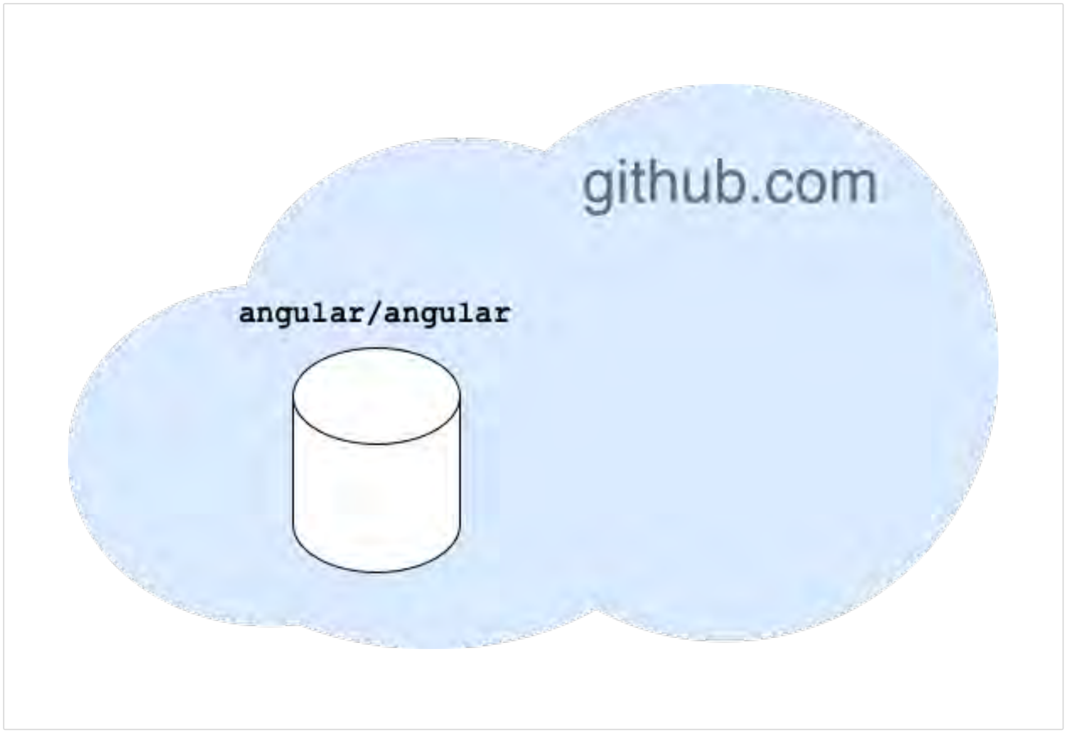
Create your repo and workspace for Angular documentation

This section describes how to create the repo and the `git` workspace necessary to edit, test, and submit changes to the Angular documentation.

> **IMPORTANT**:
> Because `git` commands are not beginner friendly, the topics in this section include procedures that should reduce the chance of `git` mishaps. Fortunately, because you are working in your own account, even if you make a mistake, you can't harm any of the Angular code or documentation.
>
> To follow the procedures in these topics, you must use the repo and directory configuration presented in this topic. The procedures in these topics are designed to work with this configuration.
>
> If you use a different configuration, the procedures in these topics might not work as expected and you could lose some of your changes.

The code and documentation for the Angular framework are stored in a public repository, or repo, on github.com in the `angular` account. The path to the Angular repo is https://github.com/angular/angular, hence the abbreviated name, `angular/angular`.

GitHub is a cloud service that hosts many accounts and repositories. You can imagine the `angular/angular` repo in GitHub as shown in this image.

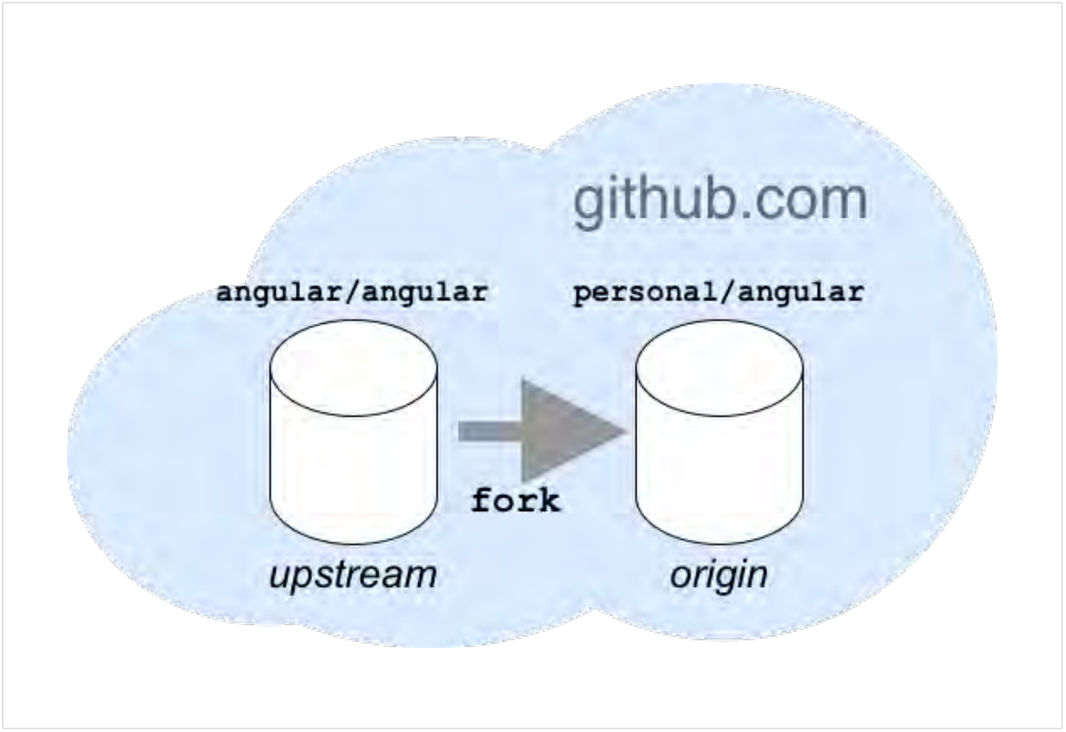


Fork the `angular/angular` repo to your account

As a public repo, `angular/angular` is available for anyone to read and copy, but not to change. While only specific accounts have permission to make changes to `angular/angular`, anyone with a GitHub account can request a change to it. Change requests to `angular/angular` are called *pull requests*. A pull request is created by one account to ask another account to pull in a change.

Before you can open a pull request, you need a forked copy of `angular/angular` in your personal GitHub account.

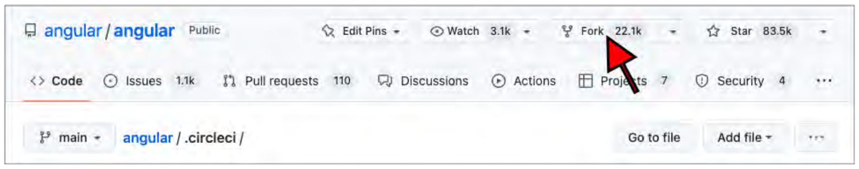
To get a forked copy of `angular/angular`, you fork the `angular/angular` repo into your personal GitHub account and end up with the repos shown in the following image. From the perspective of `personal/angular`, `angular/angular` is the upstream repo and `personal/angular` is the origin repo.



To fork the angular repo to your account

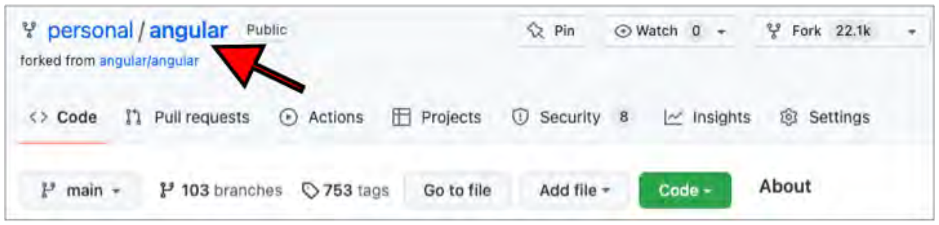
Perform this procedure in a browser.

1. Sign into your GitHub account. If you don't have a GitHub account, create a new account before you continue.
2. Navigate to `https://github.com/angular/angular`.
3. In `https://github.com/angular/angular`, click the **Fork** button near the top-right corner of the page. This image is from the top of the `https://github.com/angular/angular` page and shows the **Fork** button.



4. In **Create a new fork**:
   a. Accept the default values in **Owner** and **Repository name**.
   b. Confirm that **Copy the `main` branch only** is checked.
   c. Click **Create repository**. The forking process can take a few minutes.
5. You now have a copy of the `angular/angular` repo in your GitHub account.

After your fork of `angular/angular` is ready, your browser opens the web page of the forked repo in your GitHub account. In this image, notice that the account now shows the username of your personal GitHub account instead of the `angular` account.



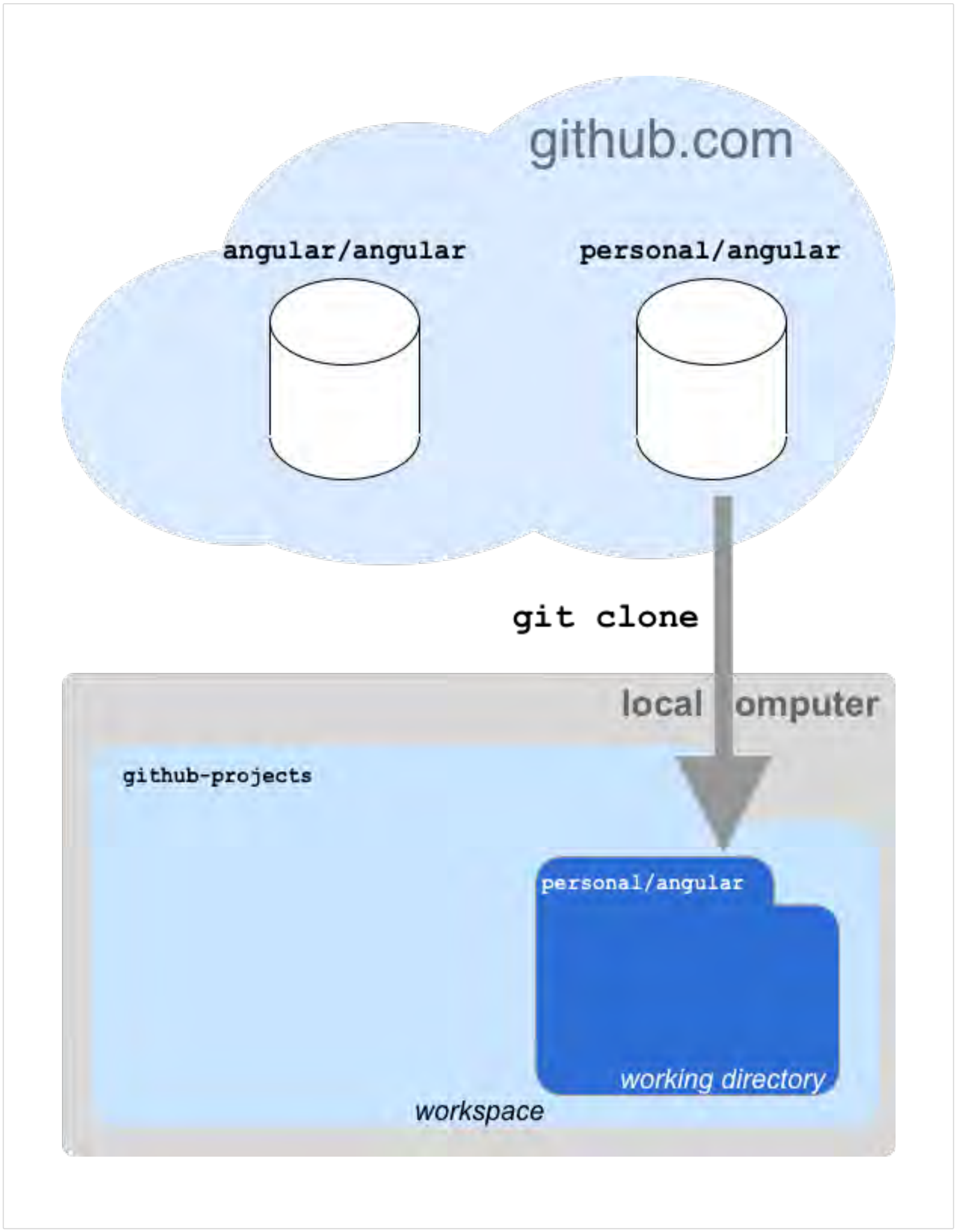
As a forked repo, your new repo maintains a reference to `angular/angular`. From your account, `git` considers your `personal/angular` repo as the origin repo and `angular/angular` as the upstream repo. You can think of this as: your changes originate in the *origin* repo and you send them *upstream* to the `angular/angular` repo. The message below the repo name in your account, `forked from angular/angular`, contains a link back to the upstream repo.

This relationship comes into play later, such as when you update your `personal/angular` repo and when you open a pull request.

Create a git workspace on your local computer

A `git` workspace on your local computer is where copies of GitHub repos in the cloud are stored on your local computer. To edit Angular documentation on your local computer, you need a clone of your origin repo, `personal/angular`.

Clone the `personal/angular` repo into the subdirectory for your account, as this illustration shows. Remember to replace `personal` with your GitHub username. The `personal/angular` directory in your workspace becomes your `working` directory. You do your editing in the working directory of your `personal/angular` repo.

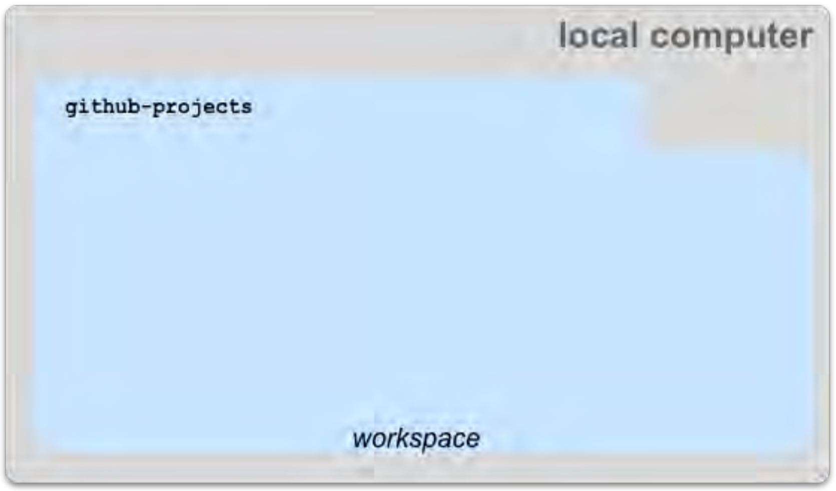


Cloning a repo duplicates the repo that's in the cloud on your local computer. There are procedures to keep the clone on your local computer in sync with the repo in the cloud that are described later.

To clone the Angular repo into your workspace

Perform these steps in a command-line tool on your local computer.

1. Navigate to the `workspace` directory. In this example, this is the directory named, `github-projects`.

   If this directory isn't on your local computer, create it, and then navigate to it before you continue.

2. From the workspace directory, run this command to create a directory for the repo from your `personal` account Remember to replace `personal` with your GitHub username.

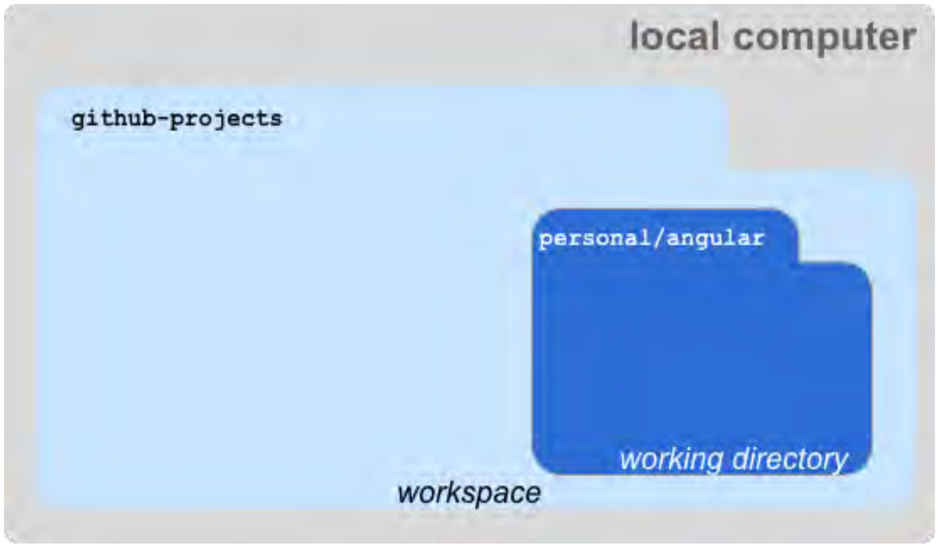
   ```
   mkdir personal
   ```

3. From the workspace directory, run this command to clone the origin `personal/angular` repo into the `personal` account directory. Remember to replace `personal` with your GitHub username.

   ```
   git clone https://github.com/personal/angular
   personal/angular
   ```

Your local computer is now configured as shown in the following illustration.

Your `working` directory is the `personal/angular` directory in your `git` workspace directory. This directory and its subdirectories have the files that you edit to fix documentation issues.



Complete the software installation

After you clone the origin repo on your local computer, run these commands from a command-line tool:

1. Install the npm modules used by the Angular project. In a command line tool on your local computer:

   a. Navigate to your `git` workspace. In this example, this is the `github-projects` directory.

   b. In your `git` workspace, run this command to navigate to the documentation root directory in your clone of the `personal/angular` repo. Remember to replace `personal` with your GitHub username.

   ```
   cd personal/angular
   ```

   c. Run this command to install the Angular dependencies.

   ```
   yarn
   ```

   d. Run this command to navigate to the documentation project.

   ```
   cd aio
   ```

   e. Run this command to install the npm modules for the documentation.

   ```
   yarn
   ```

2. Locate `angular/aio/tools/doc-linter/vale.ini` in your working directory to use in the next step as the path to the configuration file in the **Vale:Config** setting.

3. Install Vale to complete the software installation.

Build and test the Angular documentation

Angular provides tools to build and test the documentation. To review your work and before you submit your updates in a pull request, be sure to build, test, and verify your changes using these tools.

> Note that the instructions found in https://github.com/angular/angular/blob/main/docs/DEVELOPER.md are to build and test the Angular framework and not the Angular documentation.
>
> The procedures on this page build only the Angular documentation. You don't need to build the Angular framework to build the Angular documentation.

To navigate to the Angular documentation directory

Perform these steps from a command-line tool on your local computer.

1. Navigate to the Angular documentation in the working directory of your account in your `git` workspace on your local computer.
2. Navigate to your `git` workspace directory. In this example, this is the `github-projects` directory.
   a. Run this command to navigate to the working directory with the `angular` repo you forked to your personal account. Remember to replace `personal` with your GitHub username.

      ```
      cd personal/angular
      ```

   b. Run this command to navigate to the Angular documentation directory.

      ```
      cd aio
      ```

The Angular documentation directory is the root of the Angular documentation files. These directories in the `angular/aio` directory are where you find the files that are edited the most.

DIRECTORY	FILES
`angular/aio/content`	Files and other assets used in the Angular documentation
`angular/aio/content/guide`	The markdown files for most Angular documentation
`angular/aio/content/tutorial`	The markdown files used by the Tour of Heroes tutorial

The Angular documentation source has many other directories in `angular/aio` but they don't change often.

To build and view the Angular documentation on your computer

Perform these steps from a command-line tool on your local computer.

1. Build the Angular documentation.
   a. From the Angular documentation directory, run this command:
   ```
   yarn build
   ```
   b. If building the documentation reports one or more errors, fix the errors and repeat the previous step before you continue.
2. Start the local documentation server.
   a. From the documentation directory, run this command:
   ```
   yarn start
   ```
   b. Open a browser on your local computer and view your documentation at `https://localhost:4200`.
3. Review the documentation in the browser.

To run the automated tests on the Angular documentation

Perform these steps from a command-line tool on your local computer.

1. Navigate to the documentation directory, if you're not already there.
2. From the documentation directory, run this command to build the documentation before you test it:

   ```
   yarn build
   ```

3. If building the documentation returns one or more errors, fix those and build the documentation again before you continue.
4. From the documentation directory, run these commands to start the automated tests that verify the docs are consistent. These are most, but not all, of the tests that are performed after you open your pull request. Some tests can only be run in the automated testing environment.

   ```
   yarn e2e
   yarn docs-test
   ```

When you run these tests on your documentation updates, be sure to correct any errors before you open a pull request.

Next steps

After you build the documentation from your forked repo on your local computer and the tests run without error, you are ready to continue.

You have successfully configured your local computer to edit Angular documentation and open pull requests.

Continue to the other topics in this section for information about how to perform other documentation tasks.

Last reviewed on Wed Oct 12 2022

Select a documentation issue

This topic describes how to select an Angular documentation issue to fix.

Angular documentation issues are stored in the **Issues** tab of the angular/angular repo. Documentation issues can be identified by the `area: docs` label, and they are labeled by priority.

You are welcome to work on any issue that someone else isn't already working on. If you know of a problem in the documentation that hasn't been reported, you can also create a new documentation issue.

Some things to consider when choosing an issue to fix include:

- Fixing higher priority issues is more valuable to the community.
- If you're new to open source software, a lower priority issue or a `good first issue` would be a good place to start.
- Every contribution helps improve the documentation.

After you select an issue to resolve:

1. In the issue page, add `working on fix` as a comment to let others know that you are working on it.
2. Continue to Starting to edit a documentation topic.

LINKS TO DOCUMENTATION ISSUES

All open documentation issues

All open and unassigned documentation issues

Unassigned good first documentation issues

Unassigned priority 1 documentation issues

Unassigned priority 2 documentation issues

Unassigned priority 3 documentation issues

Unassigned priority 4 documentation issues

Unassigned priority 5 documentation issues

@reviewed 2022-10-12

Start to edit a documentation topic

This topic describes the tasks that you perform when you start to work on a documentation issue.

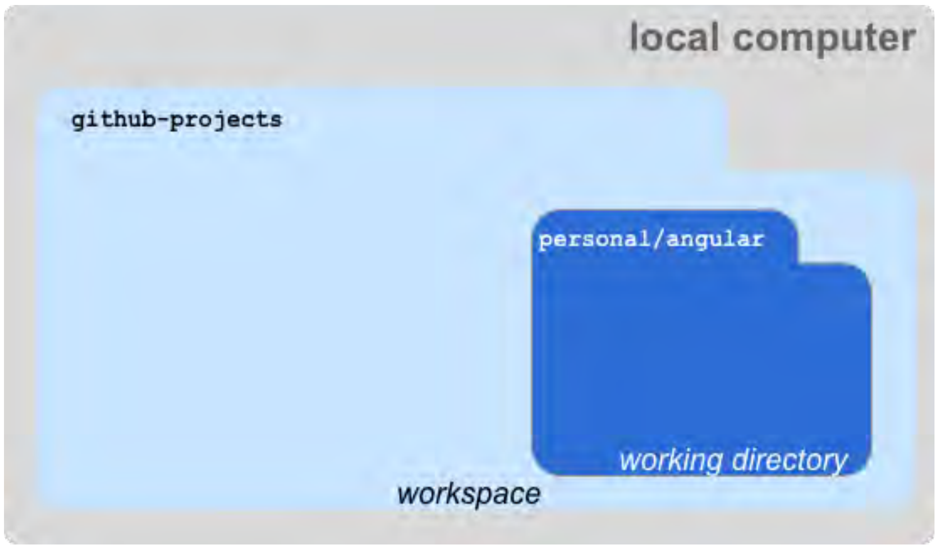
The documentation in angular.io is built from markdown source code files. The markdown source code files are stored in the `angular` repo that you forked into your GitHub account.

To update the Angular documentation, you need:

- A clone of `personal/angular`
  You created this when you created your workspace. Before you start editing a topic, update your clone of `personal/angular`.
- A `working` branch that you create from an up-to-date `main` branch. Creating your `working` branch is described later in this topic.

The procedures in this topic assume that the files on your local computer are organized as illustrated in the following diagram. On your local computer, you should have:

- Your 'git' workspace directory. In this example, the path to your 'git' workspace directory is `github-projects`.
- Your working directory, which is the directory that you created when you cloned your fork into your `git` workspace. In this example, the path to your working directory is `github-projects/personal/angular`, where `personal` is replaced with your GitHub username.



IMPORTANT:

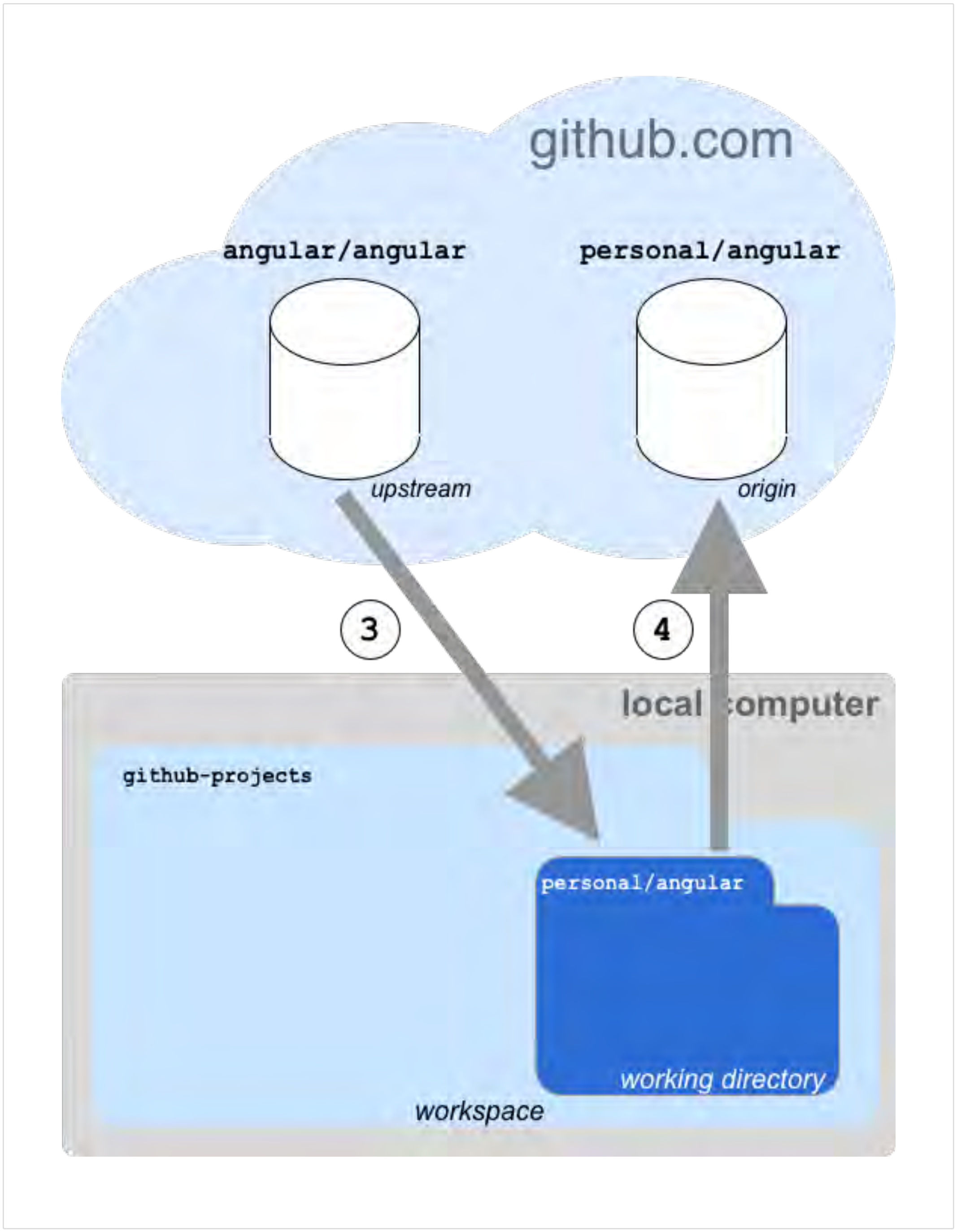
Remember to replace `personal` with your GitHub username in the commands and examples in this topic.

The procedures in this topic assume that you are starting from your workspace directory.

Update your fork with the upstream repo

Before you start editing the documentation files, you want to sync the `main` branch of your fork and its clone with the `main` branch of the upstream `angular/angular` repo.

This procedure updates the your `personal/angular` repo in the cloud and its clone on your local computer, as illustrated here. The circled numbers correspond to procedure steps.



To update your fork and its clone with the upstream repo

Perform these steps from a command-line tool on your local computer.

1. From your workspace directory, run this command to navigate to your working directory. This step is not shown in the image. Remember to replace `personal` with your GitHub username.

   ```
   cd personal/angular
   ```

2. Run this command to check out the `main` branch. This step is not shown in the image.

   ```
   git checkout main
   ```

3. Run this command to update the `main` branch in the working directory on your local computer from the upstream `angular/angular` repo.

   ```
   git fetch upstream
   git merge upstream/main
   ```

4. Run this command to update your `personal/angular` repo on `github.com` with the latest from the upstream `angular/angular` repo.

   ```
   git push
   ```

The `main` branch on your local computer is now in sync with your origin repo on `github.com`. They have been updated with any changes that have been made to the upstream `angular/angular` repo since the last time you updated your fork.

Create a working branch for editing

All your edits to the Angular documentation are made in a `working` branch in the clone of `personal/angular` on your local computer. You create the working branch from the up-to-date `main` branch of `personal/angular` on your local computer.

A working branch keeps your changes to the Angular documentation separate from the published documentation until it is ready. A working branch also keeps your edits for one issue separate from those of another issue. Finally, a working branch identifies the changes you made in the pull request that you submit when you're finished.

> **IMPORTANT**:
> Before you edit any Angular documentation, make sure that you are using the correct `working` branch. You can confirm your current branch by running `git status` from your `working` directory before you start editing.

To create a `working` branch for editing

Perform these steps in a command-line program on your local computer.

1. Update your fork of `angular/angular`.

2. From your workspace directory, run this command to navigate to your working directory. Remember to replace `personal` with your GitHub username.

   ```
   cd personal/angular
   ```

3. Run this command to check out the `main` branch.

   ```
   git checkout main
   ```

4. Run this command to create your working branch. Replace `working-branch` with the name of your working branch. Name your working branch something that relates to your editing task, for example, if you are resolving `issue #12345`, you might name the branch, `issue-12345`. If you are improving error messages, you might name it, `error-message-improvements`. A branch name can have alphanumeric characters, hyphens, underscores, and slashes, but it can't have any spaces or other special characters.

   ```
   git checkout -b working-branch
   ```

5. Run this command to make a copy of your working branch in your repo on `github.com` in the cloud. Remember to replace `working-branch` with the name of your working branch.

```
git push --set-upstream origin working-branch
```

Edit the documentation

After you create a working branch, you're ready to start editing and creating topics.

Last reviewed on Wed Oct 12 2022

Make and save changes to a documentation topic

This topic describes tasks that you perform while making changes to the documentation.

> **IMPORTANT**:
>
> Only perform these tasks after you have created a working branch in which to work as described in Create a working branch for editing.

Work in the correct working branch

Before you change any files, make sure that you are working in the correct working branch.

To set the correct working branch for editing

Perform these steps from a command-line tool on your local computer.

1. From your workspace directory, run this command to navigate to your working directory. Remember to replace `personal` with your GitHub username.

   ```
   cd personal/angular
   ```

2. Run this command to check out your working branch. Replace `working-branch` with the name of the branch that you created for the documentation issue.

   ```
   git checkout working-branch
   ```

Edit the documentation

Review the Angular documentation style guide before you start editing to understand how to write and format the text in the documentation.

In your working branch, edit the files that need to be changed. Most documentation source files are found in the `aio/content/guide` directory of the `angular` repo.

Angular development tools can render the documentation as you make your changes.

To view the rendered documentation while you are editing

Perform these steps from a command-line tool on your local computer or in the **terminal** pane of your IDE.

1. Navigate to your working directory.

2. From your working directory, run this command to navigate to the `aio` directory. The `aio` directory contains Angular's documentation files and tools.

   ```
   cd aio
   ```

3. Run this command to build the documentation locally.

   ```
   yarn build
   ```

   This builds the documentation from scratch, but does not serve it.

4. Run this command to serve and sync the documentation.

   ```
   yarn start
   ```

   This serves your draft of the angular.io website locally at `http://localhost:4200` and watches for changes to documentation files. Each time you save an update to a documentation file, the angular.io website at `http://localhost:4200` is updated. You might need to refresh your browser to see the changes after you save them.

Documentation linting

If you installed Vale on your local computer and your IDE, each time you save a markdown file, Vale reviews it for common errors. Vale, the documentation linter, reports the errors it finds in the **Problems** pane of Visual Studio Code. The errors are also reflected in the documentation source code, as close to the problem as possible.

For more information about documentation linting and resolving lint problems, see Resolve documentation linter messages.

Save your changes

As you make changes to the documentation source files on your local computer, your changes can be in one of these states.

- **Made, but not saved**

  This is the state of your changes as you edit a file in your integrated development environment (IDE).

- **Saved, but not committed**

  After you save changes to a file from the IDE, they are saved to your local computer. While the changes have been saved, they have not been recorded as a change by `git`, the version control software. Your files are typically in this state as you review your work in progress.

- **Committed, but not pushed**

  After you commit your changes to `git`, your changes are recorded as a *commit* on your local computer, but they are not saved in the cloud. This is the state of your files when you've made some progress and you want to save that progress as a commit.

- **Committed and pushed**

  After you push your commits to your personal repo in `github.com`, your changes are recorded by `git` and saved to the cloud. They are not yet part of the `angular/angular` repo. This is the state your files must be in before you can open a pull request for them to become part of the `angular/angular` repo.

- **Merged into Angular**

  After your pull request is approved and merged, the changes you made are now part of the `angular/angular` repo and appear in the angular.io web site. Your documentation update is complete.

This section describes how to save the changes you make to files in your working directory. If you are new to using `git` and GitHub, review this section carefully to understand how to save your changes as you make them.

Save your changes to your local computer

How to save changes that you make to a file on your local computer is determined by your IDE. Refer to your IDE for the specific procedure of saving changes. This process makes your changes *saved, but not committed*.

Review your rendered topics

After you save changes to a documentation topic, and before you commit those changes on your local computer, review the rendered topic in a browser.

To render your changes in a browser on your local computer

Perform these steps from a command-line tool on your local computer or in the **terminal** pane of your IDE.

1. From your workspace directory, run this command to navigate to the `aio` directory in your working directory. Remember to replace `personal` with your GitHub username.

   ```
   cd personal/angular/aio
   ```

2. Run this command to build the documentation using the files on your local computer.

   ```
   yarn build
   ```

   This command builds the documentation from scratch, but does not serve it for viewing.

3. Run this command to serve the documentation locally and rebuild it after it changes.

   ```
   yarn start
   ```

   This command serves the Angular documentation at `http://localhost:4200`. You might need to refresh the browser after the documentation is updated to see the changes in your browser.

After you are satisfied with the changes, commit them on your local computer.

Commit your changes on your local computer

Perform this procedure after you save the changes on your local computer and you are ready to commit changes on your local computer.

To commit your changes on your local computer

Perform these steps from a command-line tool on your local computer or in the **terminal** pane of your IDE.

1. From your workspace directory, run this command to navigate to the `aio` directory in your working directory. Remember to replace `personal` with your GitHub username.

   ```
   cd personal/angular/aio
   ```

2. Run this command to confirm that you are ready to commit your changes.

   ```
   git status
   ```

   The `git status` command returns an output like this.

```
On branch working-branch
Your branch is up to date with 'origin/working-
branch
Changes not staged for commit:
  (use "git add <file>..." to update what will be
committed)
  (use "git restore <file>..." to discard changes
in working directory)
        modified:   content/guide/doc-build-test.md
        modified:   content/guide/doc-edit-
finish.md
        modified:   content/guide/doc-editing.md
        modified:   content/guide/doc-pr-prep.md
        modified:   content/guide/doc-pr-update.md
        modified:   content/guide/doc-prepare-to-
edit.md
        modified:   content/guide/doc-select-
issue.md
        modified:   content/guide/doc-update-
start.md

no changes added to commit (use "git add" and/or
"git commit -a")
```

a. Confirm that you in the correct working branch.

   If you are not in the correct branch, replace `working-branch` with the name of your working branch and then run `git checkout working-branch` to select the correct branch.

b. Review the modified files in the list. Confirm that they are those that you have changed and saved, but not committed. The list of modified files varies, depending on what you have edited.

3. Run this command to add a file that you want to commit. Replace `filename` with a filename from the `git status` output.

```
git add filename
```

You can add multiple files in a single command by using wildcard characters in the filename parameter. You can also run this command to add all changed files that are being tracked by `git` to the commit by using `*` filename as this example shows.

```
git add *
```

> **IMPORTANT**:
> Files that are not tracked by `git` are not committed or pushed to your repo on `github.com`, and they do not appear in your pull request.

4. Run `git status` again.

```
git status
```

5. Review the output and confirm the files that are ready to be committed.

```
On branch working-branch
Your branch is up to date with 'origin/working-
branch'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
  modified:   content/guide/doc-build-test.md
  modified:   content/guide/doc-edit-finish.md
  modified:   content/guide/doc-editing.md
  modified:   content/guide/doc-pr-prep.md
  modified:   content/guide/doc-pr-update.md
  modified:   content/guide/doc-prepare-to-edit.md
  modified:   content/guide/doc-select-issue.md
  modified:   content/guide/doc-update-start.md
```

6. Run this command to commit the changed files to your local computer. The commit message that follows the `-m` parameter must start with `docs:` followed by space, and your message. Replace `detailed commit message` with a message that describes the changes you made.

   ```
   git commit -m 'docs: detailed commit message'
   ```

   For more information about Angular commit messages, see Formatting commit messages for a pull request.

Your changes to the documentation are now *committed, but not pushed*.

Push your commits to the cloud

After you commit the changes to your local computer, this procedure pushes those commits to your `origin` repo in the cloud.

To push your changes to your origin repo in the cloud

Perform these steps from a command-line tool on your local computer or in the **terminal** pane of your IDE.

1. From your workspace directory, run this command to navigate to the `aio` directory in your working directory. Remember to replace `personal` with your GitHub username.

   ```
   cd personal/angular/aio
   ```

2. Run this command to confirm that you are using the correct branch.

   ```
   git status
   ```

   If you aren't in the correct branch, replace `working-branch` with the name of your working branch and run `git checkout working-branch` to select the correct branch.
   Git status also shows whether you have changes on your local computer that have not been pushed to the cloud.

   ```
   On branch working-branch
   Your branch is ahead of 'origin/working-branch' by
   1 commit.
     (use "git push" to publish your local commits)
   ```

   This example output says that there is one commit on the local computer that's not in the `working-branch` branch on the `origin` repo. The `origin` is the `personal/angular` repo in GitHub. The next command pushes that commit to the `origin` repo.

3. Run this command to push the commits on your local computer to your account on GitHub in the cloud.

   ```
   git push
   ```

If this is the first time you've pushed commits from the branch, you can see a message such as this.

```
fatal: The current branch working-branch has no
upstream branch.
To push the current branch and set the remote as
upstream, use

    git push --set-upstream origin working-branch

To have this happen automatically for branches
without a tracking
upstream, see 'push.autoSetupRemote' in 'git help
config'.
```

If you get this message, copy the command that the message provides and run it as shown here:

```
git push --set-upstream origin working-branch
```

The changes that you made in the `working-branch` branch on your local computer have been saved on `github.com`. Your changes to the documentation are now *committed and pushed*.

Test your documentation

After you update the documentation to fix the issue that you picked, you are ready to test the documentation. Testing documentation consists of:

- **Documentation linting**

  Each time you open and save a documentation topic, the documentation linter checks for common errors. For more information about documentation linting, see Resolving documentation linter messages.

- **Manual review**

  When your documentation update is complete, have another person review your changes. If you have updated technical content, have a subject matter expert on the topic review your update, as well.

- **Automated testing**

  The Angular documentation is tested automatically after you open a pull request. It must pass this testing before the pull request can be merged. For more information about automated documentation testing, see Testing a documentation update.

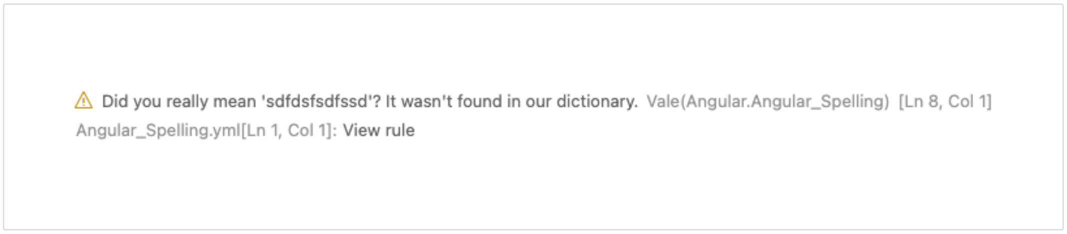
Last reviewed on Wed Oct 12 2022

Resolve documentation linter messages

This topic describes different ways to resolve common messages that the documentation linter produces.

Anatomy of a documentation linter message

This is an example of a message produced by the documentation linter.



⚠ Did you really mean 'sdfdsfsdfssd'? It wasn't found in our dictionary. Vale(Angular.Angular_Spelling) [Ln 8, Col 1]
Angular_Spelling.yml[Ln 1, Col 1]: View rule

A documentation linter message contains these elements. Starting from the top line:

- The severity. One of these icons indicates the severity of the message:
    - **Error** (A red `x` in a circle) Errors must be corrected before the file can be merged.
    - **Warning** (A yellow exclamation mark in a triangle) Warnings should be corrected before the file is merged.
    - **Info** (A blue lower-case `i` in a circle) Informational messages should be corrected before the file is merged.
- The style rule message. The style rule message in this example is:

```
Did you really mean 'sdfdsfsdfdfssd'? It wasn't
found in our dictionary.
```

- The style reference. Some references are linked to a style guide topic that explains the rule. The style reference in this example is:

```
Vale(Angular.Angular_Spelling)
```

- The location of the problem text in the document identified by source line and column as precisely as possible. Some messages might not have the exact location of the text that triggered the message. The location in this example is:

```
[Ln 8, Col 1]
```

- The style test definition file that produced the message, which is linked to the file. The style test definition in this example is:

```
Angular_Spelling.yml[Ln 1, Col 1]: View rule
```

Strategies to improve your documentation

These tips can help you improve your documentation and remove documentation linter messages.

Refer to the style guides

The lint tool tests against the styles found in these style guides. Most style tests include links to relevant sections in these documents for more information.

- Angular documentation style guide
- Google Developer Documentation Style Guide

> Not every style mentioned in the style guides has a test. Style guides and the style tests can change.

Split up long sentences

Generally, shorter sentences are easier to read than longer ones. Long sentences can occur when you try to say too much at once. Long sentences, as well as the use of parentheses, semicolons, or words identified as `too-wordy`, generally require rethinking and rewriting.

Consider restructuring a long sentence to break its individual ideas into distinct sentences or bullet points.

Use lists and tables

Sentences that contain comma-separated lists might be clearer if presented as a bulleted-list or table.

Consider changing a comma-separated list of items in a sentence to a list of bullets to make those list items easier to read.

Use more common words

Shorter, more common words are generally easier to read than longer ones. This does not mean you need to write down to the audience. Technical docs should still be precise. Angular docs are read by many people around the world and should use language that the most people can understand.

If you think a specific term is required even though it has been flagged as uncommon, try to include a short explanation of the term. Also, try adding some context around its first mention.

Linking a term to another section or topic is also an option, but consider the disruption that causes to the reader before you use it. If you force a reader to go to another page for a definition, they might lose their concentration on the current topic and their primary goal.

Use fewer words

If you can remove a word and not lose the meaning of the sentence, leave it out.

One common place where removing words can help is in a list of examples with more than two or three items. Before you place the items in a bullet list, consider if only one of the items can convey the desired meaning. Another option might be to replace a list of items with a single term that describes all the elements in your list.

More about specific documentation linter messages

Most documentation linter messages are self-explanatory and include a link to supplementary documentation. Some messages identify areas in that the documentation might need more thought. The following types of messages often occur in areas of the text that should be reconsidered and rewritten to improve the text and remove the message.

A word is `too-wordy` or should be replaced by

another

Generally, technical documentation should use a simple and consistent vocabulary to be understood by a wide audience. Words that trigger this message are usually words for which there's a simpler way to convey the same thought.

Angular.WriteGood_TooWordy - See if you can rewrite the sentence...

Words identified by this style test can usually be replaced by simpler words. If not, sentences with these words should be revised to use simpler language and avoid the word in the message.

The following table has some common words detected by this type of message and simpler words to try in their place.

`TOO-WORDY` WORD	SIMPLER REPLACEMENT
`accelerate`	`speed up`
`accomplish`	`perform` or `finish`
`acquire`	`get`
`additional`	`more`
`adjustment`	`change`
`advantageous`	`beneficial`
`consequently`	`as a result`
`designate`	`assign`
`equivalent`	`the same`
`exclusively`	`only`
`for the most part`	`generally`
`have a tendency to`	`tend to`
`in addition`	`furthermore`
`modify`	`change` or `update`
`monitor`	`observe`
`necessitate`	`require`

`TOO-WORDY` WORD	SIMPLER REPLACEMENT
`one particular`	`one`
`point in time`	`moment`
`portion`	`part`
`similar to`	`like`
`validate`	`verify`
`whether or not`	`whether`

`WordList` messages

The messages about words detected by these style tests generally suggest a better alternative. While the word you used would probably be understood, it most likely triggered this message for one of the following reasons:

- The suggested works better in a screen-reader context
- The word that you used could produce an unpleasant response in the reader
- The suggested word is simpler, shorter, or easier for more people to understand
- The word you used has other possible variations. The suggested word is the variation to use in the documentation to be consistent.

`Proselint` messages

The Proselint style tests test for words that are jargon or that could be offensive to some people.

Rewrite the text to replace the jargon or offensive language with more inclusive language.

`Starting a sentence` messages

Some words, such as *so* and *there is/are*, aren't necessary at the beginning of a sentence. Sentences that start with the words identified by this message can usually be made shorter, simpler, and clearer by rewriting them without those openings.

Cliches

Cliches should be replaced by more literal text.

Cliches make it difficult for people who don't understand English to understand the documentation. When cliches are translated by online tools such as Google Translate, they can produce confusing results.

If all else fails

The style rules generally guide you in the direction of clearer content, but sometimes you might need to break the rules. If you decide that the best choice for the text conflicts with the linter, mark the text as an exception to linting.

The documentation linter checks only the content that is rendered as text. It does not test code-formatted text. One common source of false problems is code references that are not formatted as code.

If you use these exceptions, please limit the amount of text that you exclude from analysis to the fewest lines possible.

When necessary, you can apply these exceptions to your content.

1. **General exception**

   A general exception allows you to exclude the specified text from all lint testing.

   To apply a general exception, surround the text that you do not want the linter to test with the HTML `comment` elements shown in this example.

   ```
   <!-- vale off -->

   Text the linter does not check for any style
   problem.

   <!-- vale on -->
   ```

   Be sure to leave a blank line before and after each comment.

2. **Style exception**

   A style exception allows you to exclude text from an individual style test.

   To apply a style exception, surround the text that you do not want the linter to test with these HTML `comment` elements. Between these comments, the linter ignores the style test in the comment, but still tests for all other styles that are in use.

   ```
   <!-- vale Style.Rule = NO -->
   <!-- vale Style.Rule = YES -->
   ```

   Replace `Style.Rule` in the comments with the style rule reference from the problem message displayed in the IDE. For example, imagine that you got this problem message and you want to use the word it identified as a problem.

```
Did you really mean 'inlines'?  It was not found in
our dictionary. Vale(Angular.Angular_Spelling) [Ln
24, Col 59]
Angular_Spelling.yml[Ln 1, Col 1]: View rule
```

The `Style.Rule` for this message is the text inside the parentheses: `Angular.Angular_Spelling` in this case. To turn off that style test, use the comments shown in this example.

```
<!-- vale Angular.Angular_Spelling = NO -->

'inlines' does not display a problem because this
text is not spell-checked.
Remember that the linter does not check any
spelling in this block of text.
The linter continues to test all other style rules.

<!-- vale Angular.Angular_Spelling = YES -->
```

Last reviewed on Wed Oct 12 2022

Build and test a documentation update

After you have completed your documentation update, you want to run the documentation's end-to-end tests on your local computer. These tests are some of the tests that are run after you open a pull request. You can find end-to-end test failures faster when you run them on your local computer than after you open a pull request.

Build the documentation on your local computer

Before you test your updated documentation, you want to build it to make sure you test your latest changes.

To build the documentation on your local computer

Perform these steps from a command-line tool on your local computer or in the **terminal** pane of your IDE.

1. Navigate to your working directory.

2. From your working directory, run this command to navigate to the `aio` directory. The `aio` directory contains Angular's documentation files and tools.

   ```
   cd aio
   ```

3. Run this command to build the documentation locally.

   ```
   yarn build
   ```

   This builds the documentation from scratch.

After you build the documentation on your local computer, you can run the angular.io end-to-end test.

Run the angular.io end-to-end test on your local computer

This procedure runs most, but not all, of the tests that are run after you open a pull request.

To run the angular.io end-to-end test on your local computer

On your local computer, in a command line tool or the **Terminal** pane of your IDE:

1. Run this command from your workspace directory to navigate to your working directory.

   ```
   cd personal/angular
   ```

2. Replace `working-branch` with the name of your `working` branch and run this command to check out your `working` branch.

   ```
   git checkout working-branch
   ```

3. Run this command to navigate to the documentation.

   ```
   cd aio
   ```

4. Run these commands to run the end-to-end tests.

   ```
   yarn e2e
   yarn docs-test
   ```

5. Watch for errors that the test might report.

No errors reported

If the end-to-end tests report no errors and your update has passed all other reviews required, your documentation update is ready for a pull request.

After you open your pull request, GitHub tests the code in your pull request. The tests that GitHub runs include the end-to-end tests that you just ran and other tests that only run in the GitHub repo. Because of that, even though your update passed the end-to-end tests locally, your update could still report an error after you open a pull request.

Errors reported

If the end-to-end tests report an error on your local computer, be sure to correct it before you open a pull request. If the update fails the end-to-end test locally, it is likely to also fail the tests that run after you open a pull request.

Last reviewed on Wed Oct 12 2022

Prepare a documentation update for a pull request

This topic describes how to prepare your update to the Angular documentation so that you can open a pull request. A pull request is how you share your update in a way that allows it to be merged it into the `angular/angular` repo.

> **IMPORTANT**:
>
> Make sure that you have reviewed your documentation update, removed any lint errors, and confirmed that it passes the end-to-end (e2e) tests without errors.

A pull request shares a branch in `personal/angular`, your forked repo, with the `angular/angular` repo. After your pull request is approved and merged, the new commits from your branch are added to the main branch in the `angular/angular` repo. The commits in your branch, and their messages, become part of the `angular/angular` repo.

What does this mean for your pull request?

1. Your commit messages become part of the documentation of the changes made to Angular. Because they become part of the `angular/angular` repo, they must conform to a specific format so that they are easy to read. If they aren't correctly formatted, you can fix that before you open your pull request.

2. You might need to squash the commits that you made while developing your update. It's normal to save your changes as intermediate commits while you're developing a large update, but your pull request represents only one change to the `angular/angular` repo. Squashing the commits from your working branch into fewer, or just one commit, makes the commits in your pull request match the changes your update makes to the `angular/angular` repo.

Format commit messages for a pull request

Commits merged to `angular/angular` must have messages that are correctly formatted. This section describes how to correctly format commit messages.

Remember that the commit message is different from the pull request comment.

Single line commit messages

The simplest commit message is a single-line of text. All commit messages in a pull request that updates documentation must begin with `docs:` and be followed by a short description of the change.

The following is an example a valid Angular commit message.

```
docs: a short summary in present tense without
capitalization or ending period
```

This is an example of a commit command with the single-line commit message from the previous example.

```
git commit -m "docs: a short summary in present tense
without capitalization or ending period"
```

Multi-line commit messages

You can include more information by providing a more detailed, multi-line message. The detailed body text of the message must be separated by a blank line after the summary. The footer that lists the issue the commit fixes must also be separated from the body text by a blank line.

```
docs: a short summary in present tense without
capitalization or ending period

A description of what was fixed, and why.
This description can be as detailed as necessary and
can be written with
appropriate capitalization and punctuation

Fixes #34353
```

This is an example of a commit command with a multi-line commit message from the previous example.

```
git commit -m "docs: a short summary in present tense
without capitalization or ending period

A description of what was fixed, and why.
This description can be as detailed as necessary and
can be
written with appropriate capitalization and
punctuation.

Fixes #34353"
```

This example is for documentation updates only. For the complete specification of Angular commit messages, see Commit message format.

Change a commit message

If the last commit you made has a message that isn't in the correct format, you can update the message. Changing the message of an earlier commit or of multiple commits is also possible, but requires a more complex procedure.

Run this command to change the commit message of the most recent commit. The new commit message is formatted as described in the previous procedures.

```
git commit --amend -m "New commit message"
```

This command creates a new commit on your local computer that replaces the previous commit. You must push this new commit before you open your pull request. If you pushed the original commit to the repo in your GitHub account, run this command to force-push the commit with the new message.

```
git push --force-with-lease
```

If you haven't pushed the commit you amended, you can run `git push` with no parameters to push your updated commit.

Prepare your branch for a pull request

When you created your working branch to update the documentation, you branched off the `main` branch. Your changes in the working branch were based on the state of the `main` branch at that time you created the branch.

Since you created your working branch, it's quite likely that the `main` branch has been updated. To make sure that your updates work with the current `main` branch, you should `rebase` your working branch to catch it up to what is current. You might also need to squash the commits you made in your working branch to combine them for the pull request.

Rebase your working branch

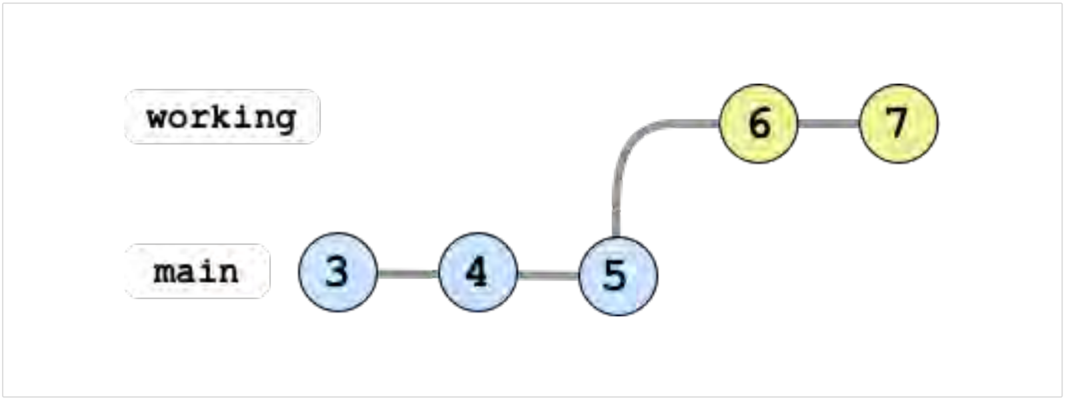
Rebasing your working branch changes the starting point of your commits from where the `main` branch was when you started to where it is now.

Before you can rebase your working branch, you must update both your *clone* and your *fork* of the upstream repo.

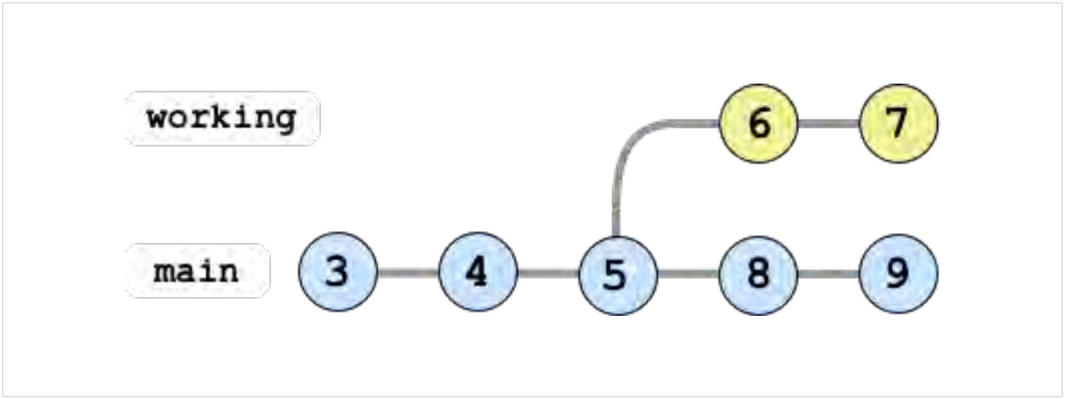
Why you rebase your working branch

Rebasing your working branch to the current state of the `main` branch eliminates conflicts before your working branch is merged back into `main`. By rebasing your working branch, the commits in your working branch show only those changes that you made to fix the issue. If you don't rebase your working branch, it can have merge commits. Merge commits are commits that `git` creates to make up for the changes in the `main` branch since the `working` branch was created. Merge commits aren't harmful, but they can complicate a future review of the changes. The following illustrates the rebase process.

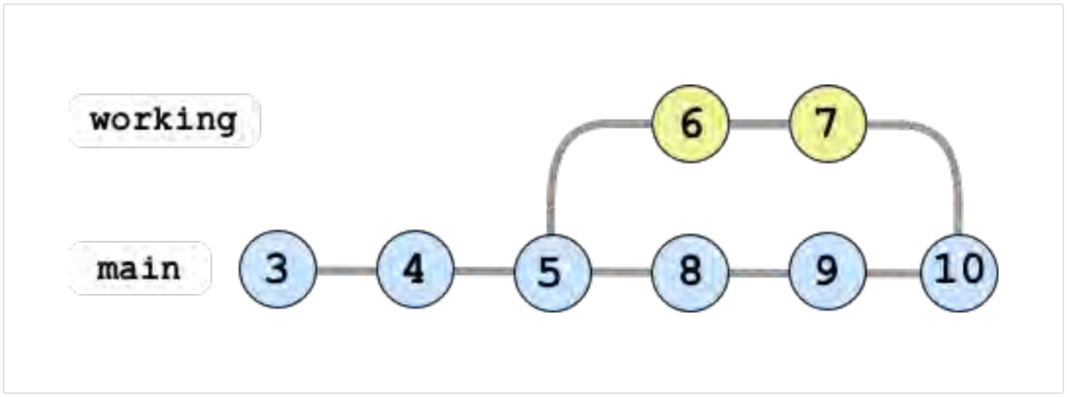
This image shows a `working` branch created from commit 5 of the `main` branch and then updated twice. The numbered circles in these diagrams represent commits.



This image shows the `main` branch after it was updated twice as the `working` branch was updated.



If the working branch was merged, a merge commit would be needed. This image illustrates the result.

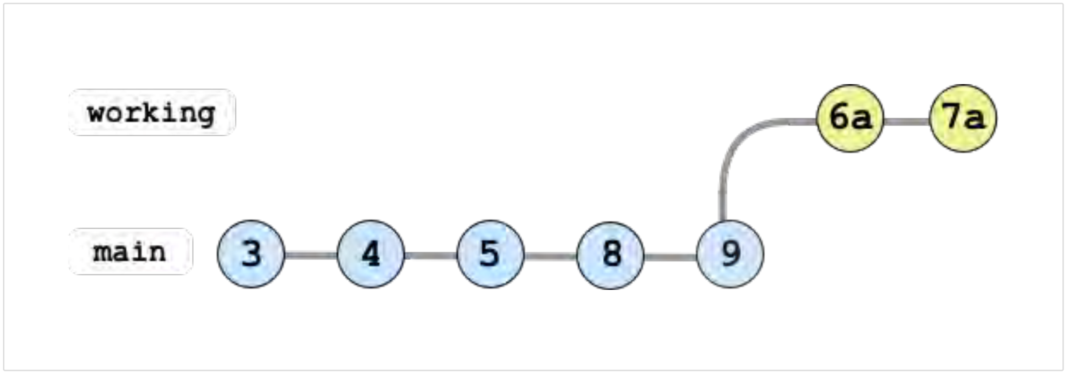


To make it easy for future contributors, the Angular team tries to keep the commit log as a linear sequence of changes. Incorporating merge commits includes changes that are the result of the merge along with what the author or developer changed. This makes it harder for future developers and authors to tell how the content evolved.

To create a linear sequence of changes, you might need to update your `working` branch and update your changes. To add your updates to the current state of the `main` branch and prevent a merge commit, you rebase the `working` branch.

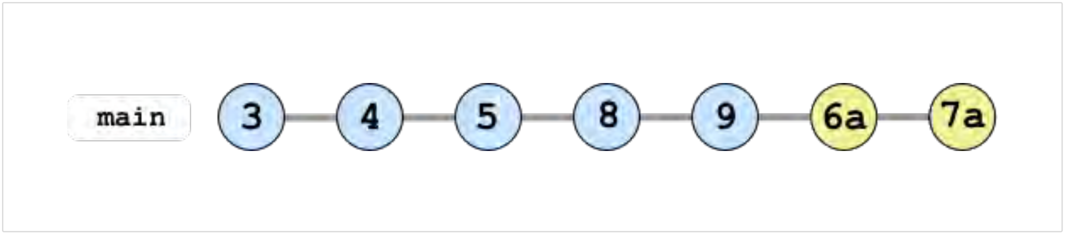
Rebasing is how `git` updates your working branch to make it look like you created it from commit `9`. To do this, it updates the commits in the `working` branch. After rebasing the `working` branch, its commits now start from the last commit of the `main` branch.

This image shows the rebased `working` branch with is updated commits.



When the rebased `working` branch is merged to main, its commits can now be appended to the `main` branch with no extra merge commits.

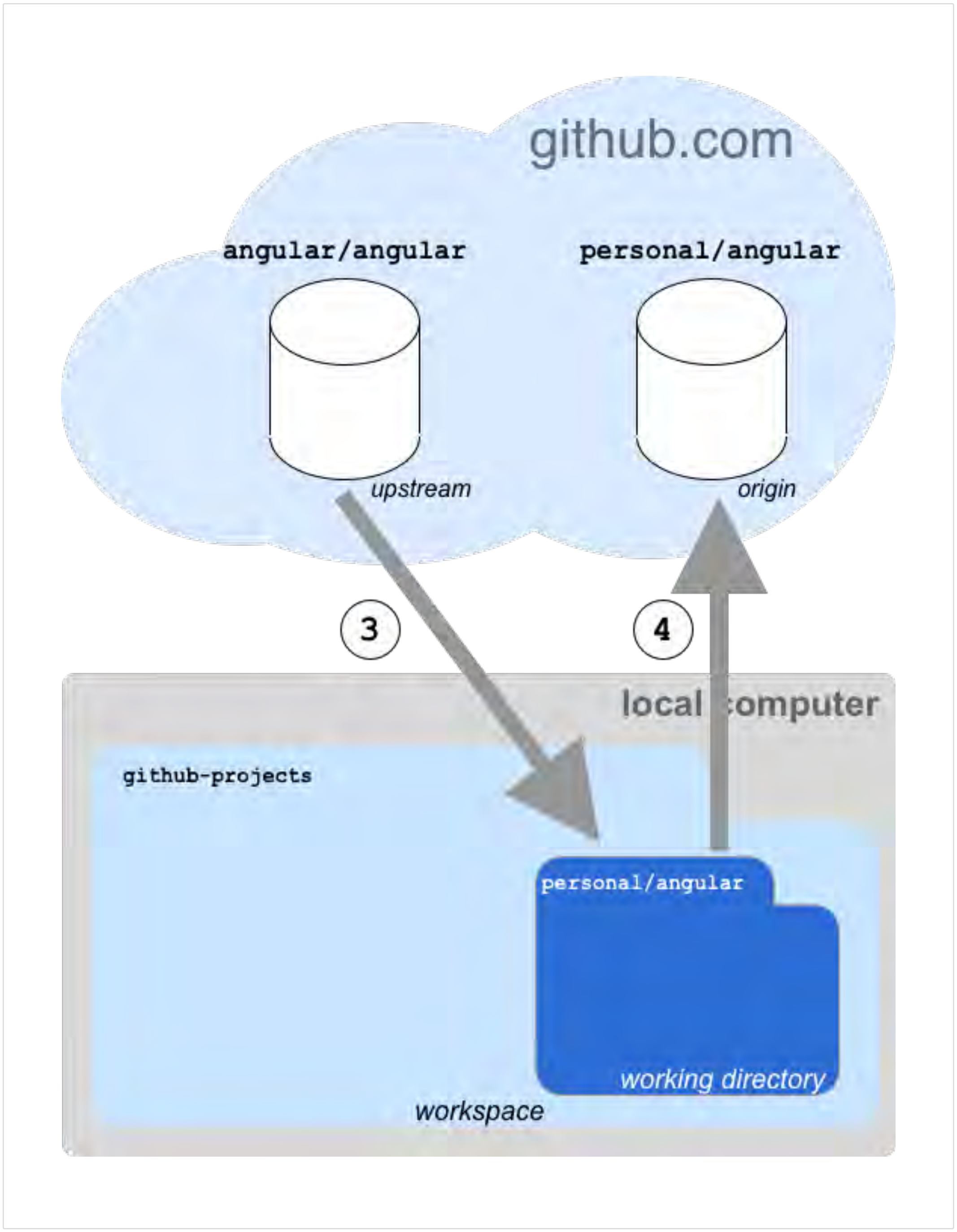
This image shows the linear, `main` branch after merging the updated and rebased `working` branch.



To update your fork of the upstream repo

You want to sync the `main` branch of your origin repo with the `main` branch of the upstream `angular/angular` before you open a pull request.

This procedure updates your origin repo, the `personal/angular` repo, on your local computer so it has the current code, as illustrated here. The circled numbers correspond to procedure steps. The last step of this procedure then pushes the update to the fork of the `angular` in your GitHub account.



Perform these steps from a command-line tool on your local computer.

1. From your workspace directory, run this command to navigate to your working directory. Remember to replace `personal` with your GitHub username.

   ```
   cd personal/angular
   ```

2. Run this command to check out the `main` branch.

   ```
   git checkout main
   ```

3. Update the `main` branch in the `working` directory on your local computer from the upstream `angular/angular` repo.

   ```
   git fetch upstream
   git merge upstream/main
   ```

4. Update your `personal/angular` repo on `github.com` with the latest from the upstream `angular/angular` repo.

   ```
   git push
   ```

The `main` branch on your local computer and your origin repo on `github.com` are now in sync. They have been updated with any changes to the upstream `angular/angular` repo that were made since the last time you updated your fork.

To rebase your working branch

Perform these steps from a command-line tool on your local computer.

1. From your workspace directory, run this command to navigate to your working directory. Remember to replace `personal` with your GitHub username.

   ```
   cd personal/angular
   ```

2. Run this command to check out your `working` branch. Replace `working-branch` with the name of your `working` branch.

   ```
   git checkout working-branch
   ```

3. Run this command to rebase your branch to add the commits from your `working` branch to the current content in the `main` branch.

   ```
   git rebase main
   ```

4. Run this command to update your `working` branch in the repo in your GitHub account.

   ```
   git push --force-with-lease
   ```

Review the commits in your working branch

After you rebase your `working` branch, your commits should be after those of the current `main` branch.

To review the commits that you've added to the `working` branch

Perform these steps from a command-line tool on your local computer.

1. From your workspace directory, run this command to navigate to your working directory. Remember to replace `personal` with your GitHub username.

   ```
   cd personal/angular
   ```

2. Run this command to confirm that you are using the correct `working` branch. If you aren't in the correct branch, replace `working-branch` with the name of your `working` branch and run `git checkout working-branch` to select the correct branch.

   ```
   git status
   ```

3. Review the message from the previous `git status` command. If you aren't in the correct branch, replace `working-branch` with the name of your `working` branch and run `git checkout working-branch` to select the correct branch.

4. Run this command to get a list of the commits in your `working` branch.

   ```
   git log --pretty=format:"%h %as %an %Cblue%s
   %Cgreen%D"
   ```

   This command returns the log of commits in the `working` branch with the most recent commit at the top of the list.

5. In the output of the previous `git log` command, find the entry that contains `upstream/main`. It should be near the top of the list.

a. **Confirm that the entry that contains `upstream/main` also contains `origin/main` and `main`**
   If it doesn't, you must resync your clone and your fork of `angular/angular`, and then rebase the branch before you continue.

b. **Confirm that all commits for your update are after the entry that contains `upstream/main`**
   Remember that the log output is displayed with the most recent commit first. Your commits should all be on top of the entry that contains `upstream/main` in the log output. If any of your commits are listed after the entry that contains `upstream/main`, somehow your commits in the `working` branch got mixed up. You must fix the branch before you open a pull request.

c. **Confirm that your commit messages are in the correct format**
   The commit message format is tested by the automated tests and it must be in the correct format before the pull request can be approved.

d. **Confirm that your commits and their messages reflect the changes your update makes to Angular**
   If you have more commits than changes, you might need to squash them into fewer commits before your pull request is approved.

Next step

After you confirm that your updates and your `working` branch are correct, you are ready to open a pull request.

Last reviewed on Wed Oct 12 2022

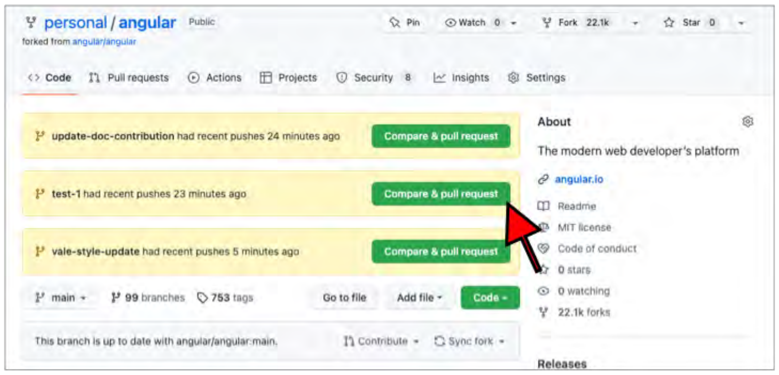
Open a documentation pull request

This topic describes how to open the pull request that requests your documentation update to be added to the `angular/angular` repo.

These steps are performed in your web browser.

1. Locate the `working` branch that you want to use for your pull request. In this example, `test-1` is the name of the `working` branch. Choose one of these options to open a pull request.

a. If you recently pushed the branch that you want to use to the `origin` repo, you might see it listed on the code page of the `angular` repo in your GitHub account. This image shows an example of a repo that has had several recent updates.



update-doc-contribution had recent pushes 24 minutes ago

test-1 had recent pushes 23 minutes ago

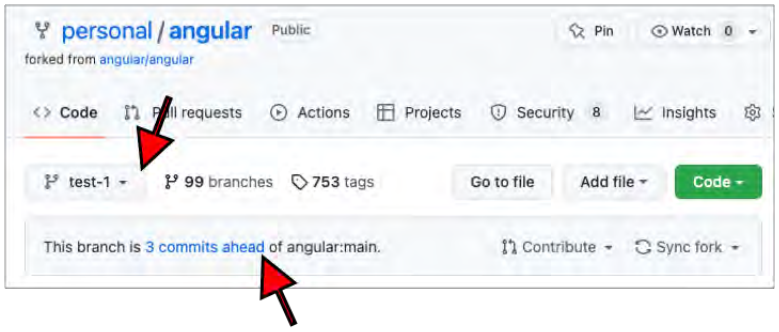
vale-style-update had recent pushes 5 minutes ago

This branch is up to date with angular/angular:main.

The modern web developer's platform

In the alert message with your `working` branch, click **Compare & pull request** to open a pull request and continue to the next step.

b. You can also select your `working` branch in the code page of the origin repo.

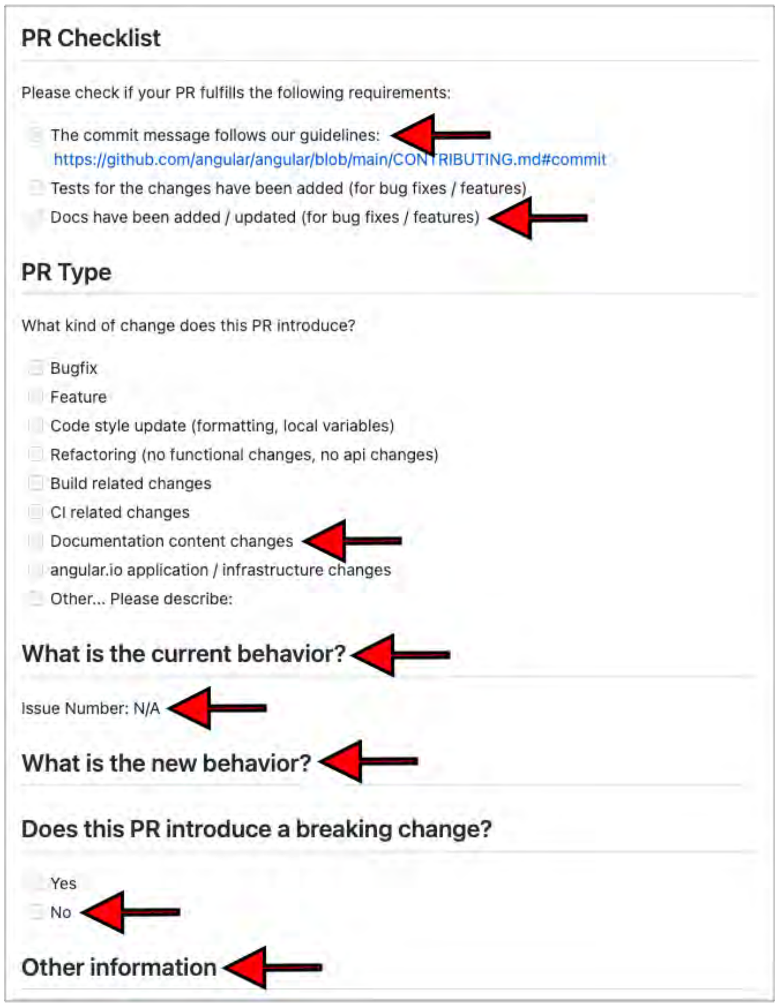


This branch is 3 commits ahead of angular:main.

Click the link text in the `"This branch is"` message to open the **Comparing changes** page.

In the **Comparing changes** page, click **Create pull request** to open the new pull request page.

c. Review and complete the form in the comment field. Most documentation updates require responses to the entries noted by an arrow and described below.



PR Checklist

Please check if your PR fulfills the following requirements:

- The commit message follows our guidelines: https://github.com/angular/angular/blob/main/CONTRIBUTING.md#commit
- Tests for the changes have been added (for bug fixes / features)
- Docs have been added / updated (for bug fixes / features)

PR Type

What kind of change does this PR introduce?

- Bugfix
- Feature
- Code style update (formatting, local variables)
- Refactoring (no functional changes, no api changes)
- Build related changes
- CI related changes
- Documentation content changes
- angular.io application / infrastructure changes
- Other... Please describe:

What is the current behavior?

Issue Number: N/A

What is the new behavior?

Does this PR introduce a breaking change?

- Yes
- No

Other information

1. **The commit message follows our guidelines**

   Mark this comment when you're sure your commit messages are in the correct format. Remember that the commit messages and the pull request title are different. For more information about commit message formatting, see Preparing a documentation update for a pull request and Commit message format.

2. **Docs have been added / updated (for bug fixes / features)**

   Mark this comment to show that documentation has been updated.

3. **Documentation content changes**

   Mark this comment to identify this is a documentation pull request. If you also updated other types of content, you can mark those as well.

4. **What is the current behavior?**

   Briefly describe what wasn't working or what was incorrect in the documentation before you made the changes in this pull request. Add the issue number here, if the problem is described in an issue.

5. **What is the new behavior?**

   Briefly describe what was added to fix the problem.

6. **Does this PR introduce a breaking change?**

   For most documentation updates, the answer to this should be `No`.

7. **Other information**

   Add any other information that can help reviewers understand your pull request here.

1. Click the arrow next to **Draft pull request** and select whether you want to create a draft pull request or a pull request.
    a. A draft pull request runs the continuous integration (CI) testing, but does not send the pull request to reviewers. You can ask people to review it by sending them the pull request link. You might use this option to see how your pull request passes the CI testing before you send it for review to be merged. Draft pull requests cannot be merged.
    b. A pull request runs the continuous integration (CI) testing and sends your pull request to reviewers to review and merge.

    > **NOTE**:
    > You can change draft pull requests to pull requests.

2. Click **Create the pull request** or **Draft pull request** to open the pull request. After GitHub creates the pull request, the browser opens the new pull request page.
3. After you open the pull request, the automated tests start running.

What happens after you open a pull request

In most cases, documentation pull requests that pass the automated tests are approved within a few days.

Sometimes, reviewers suggest changes for you to make to improve your pull request. In those case, review the suggestions and update the pull request with a comment or an updated file.

What happens to abandoned pull requests

While it can take a few days to respond to comments, try to respond as quickly as you can. Pull requests that appear to abandoned or ignored are closed according to this schedule:

- After 14 days of inactivity after the last comment, the author is reminded that the pull request has pending comments
- After 28 days of inactivity after the last comment, the pull request is closed and not merged

Last reviewed on Wed Oct 12 2022

Update a documentation pull request

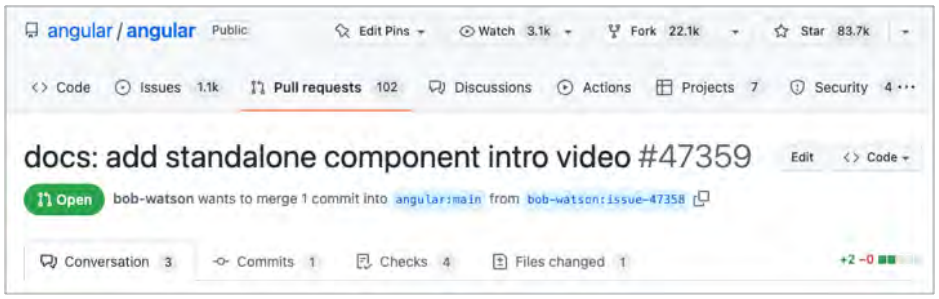
This topic describes how to respond to test failures and feedback on your pull request.

After you open a pull request, it is tested and reviewed. After it's approved, the changes are merged into `angular/angular` and they become part of the Angular documentation.

While some pull requests are approved with no further action on your part, most pull requests receive feedback that requires you to make a change.

Anatomy of a pull request

After you open a pull request, the pull request page records the activity on the pull request as it is reviewed, updated, and approved. This is an example of the top of a pull request page followed by a description of the information it contains.



docs: add standalone component intro video #47359

bob-watson wants to merge 1 commit into angular:main from bob-watson:issue-47358

Above the pull-request tabs is a summary of the pull request that includes:

- The pull request title and index
- The status of the pull request: open or closed
- A description of the branch with the changes and the branch to update

The tabs contain different aspects of the pull request.

- **Conversation**

  All comments and changes to the pull request, system messages, and a summary of the automated tests and approvals.

- **Commits**

  The log of the commits included in this pull request.

- **Checks**

  The results of the checks run on the commit. This is different from the automated tests that are also run and summarized at the bottom of the **Conversation** tab.

- **Files changed**

  The changes this request makes to the code. In this tab is where you find specific comments to the changes in your pull request. You can reply to those comments in this tab, as well.

Respond to a comment

If your pull request receives comments from a reviewer, you can respond in several ways.

- Reply to the feedback.
  For example, you can ask for more information or reply with an explanation.
- Make the changes to the documentation that the reviewer recommends.
  If you update the working branch with the suggested changes, resolve the comment.
- Make other changes to the documentation.
  After reviewing the feedback, you might see an even better improvement. Update the working branch with your improvement and explain why you chose that to your reviewers in a comment.

Remember that pull requests that do not receive a response to a review comment are considered abandoned and closed. For more information about abandoned pull requests, see What happens to abandoned pull requests.

Update a file in the pull request

Follow this procedure to change a file in the pull request or to add a new file to the pull request.

1. In your `git` workspace, in your working directory, checkout your working branch.

2. Update the documentation to respond to the feedback you received. The procedures used to revise a documentation topic are also used to update the documentation while there's an open pull request.

3. Test your update locally as described in Testing a documentation update.

4. After your updates have been tested, commit your changes and push the new commits to the working branch of your repo on your `origin` server.

5. After you update the working branch on your `origin` server, the fork of the `angular/angular` repo in your GitHub account, your pull request updates automatically.

6. After the pull request updates, the automated tests are restarted and the reviewers are notified.

Repeat this process as needed to address the feedback you get from reviews of the pull request.

Clean up the branch

If you added commits to address review feedback, you might be requested to clean up your working branch. If some of the commits you made address only review feedback from your reviewers, they can probably be squashed. Squashing commits combines the changes made in multiple commits into a single commit.

To squash commits in your working branch

Perform these steps from a command-line tool on your local computer.

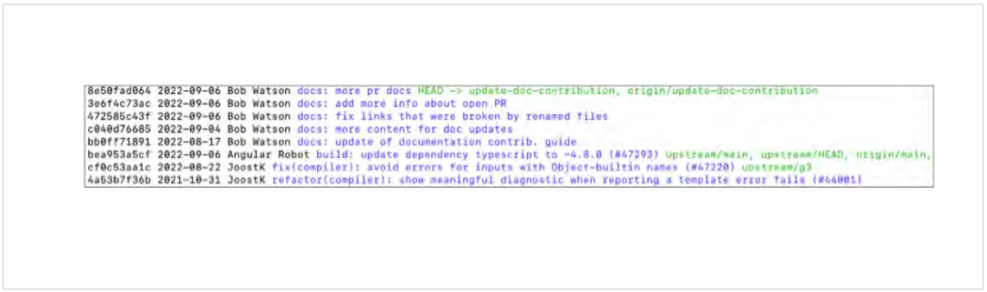
1. In your workspace directory, in your working directory, checkout your working branch.
2. Run this command to view the commits in your working branch.

   ```
   git log --pretty=format:"%h %as %an %Cblue%s
   %Cgreen%D"
   ```

3. In the output of the previous `git log` command, find the entry that contains `upstream/main`. It should be near the top of the list.
   a. **Confirm that entry also contains `origin/main` and `main`**
   If it doesn't, you must resync the clone on your local computer and your `personal/angular` repo with the `upstream` repo. After you resync the repos, rebase the working branch before you continue.

   b. **Confirm that all commits for your update are after the entry that contains `upstream/main`**
   Remember that the log output is displayed with the most recent commit first. Your commits should all be on top of the entry that contains `upstream/main` in the log output. If you have commits that are listed after the entry that contains `upstream/main`, somehow your commits in the working branch got mixed up. You must fix the branch before you try to squash any commits.

4. Count the lines that are on top of the entry that contains `upstream/main`. For example, in this log output, the working branch name is `update-doc-contribution` and there are five commit entries that are on top of the entry that contains `upstream/main`.

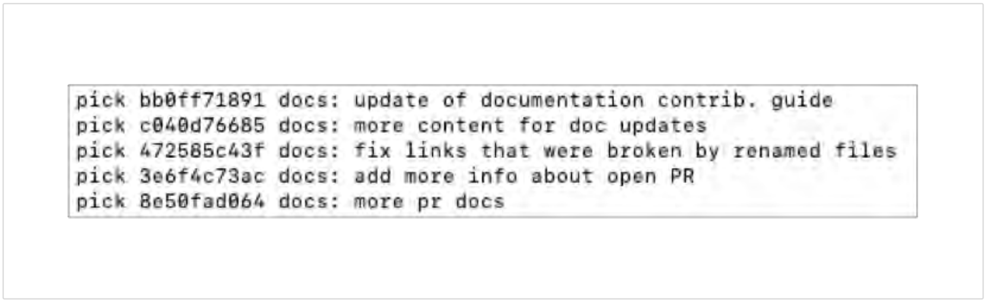


```
8e50fad064 2022-09-06 Bob Watson docs: more pr docs HEAD -> update-doc-contribution, origin/update-doc-contribution
3e6f4c73ac 2022-09-06 Bob Watson docs: add more info about open PR
472585c43f 2022-09-06 Bob Watson docs: fix links that were broken by renamed files
c040d76685 2022-09-04 Bob Watson docs: more content for doc updates
bb0ff71891 2022-08-17 Bob Watson docs: update of documentation contrib. guide
bea953a5cf 2022-09-06 Angular Robot build: update dependency typescript to ~4.8.0 (#47293) upstream/main, upstream/HEAD, origin/main,
cf0c53aa1c 2022-08-22 JoostK fix(compiler): avoid errors for inputs with Object-builtin names (#47220) upstream/g3
4a53b7f36b 2021-10-31 JoostK refactor(compiler): show meaningful diagnostic when reporting a template error fails (#44001)
```

5. Run this command to squash the commits that occurred after the entry that contains `upstream/main`. In your command, replace the `5` after `HEAD` with the number of commits on top of the entry that contains `upstream/main`.

```
git rebase -i HEAD~5
```

6. This command opens your default editor with entries for the commits that you selected in the `git rebase` command.



```
pick bb0ff71891 docs: update of documentation contrib. guide
pick c040d76685 docs: more content for doc updates
pick 472585c43f docs: fix links that were broken by renamed files
pick 3e6f4c73ac docs: add more info about open PR
pick 8e50fad064 docs: more pr docs
```

7. To squash the commits, edit the commands in the file that's presented in the editor. The commands in the editor are listed from oldest to newest, which is the opposite order from how they are listed by the `git log` command. The possible command options are listed in the editor below the commands. To squash the commits for your pull request, you only need: `pick` and `squash`.

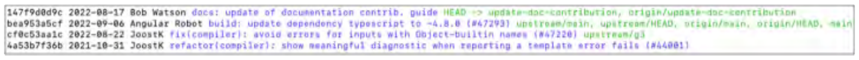
8. Review the commands in the editor and change them to match your intention.

The commands are processed from top to bottom, that is from oldest commit to the most recent.\ To merge all commits in this branch for this pull request, change the `pick` commands to `squash` for all commits except for the first one. This text shows how this looks for this example.

```
pick bb0ff71891 docs: update of documentation
contrib. guide
squash c040d76685 docs: more content for doc
updates
squash 472585c43f docs: fix links that were broken
by renamed files
squash 3e6f4c73ac docs: add more info about open PR
squash 8e50fad064 docs: more pr docs
```

With this edit, `git rebase` picks the first commit and combines the later commits into the first one.
The commit message of the commit with the `pick` command, is the commit message used for the resulting commit. Make sure that it in the correct format and starts with `docs:`. If you need to change the commit message, you can edit it in the editor.

9. After you update the commands, save and exit the editor. The `git rebase` commit processes the commands and updates the commit log in your workspace. In this example, the rebase command combined the five commits to create a single commit in your working branch. This is the commit log after the rebase command completes.



```
147f9d0d9c 2022-08-17 Bob Watson docs: update of documentation contrib. guide HEAD -> update-doc-contribution, origin/update-doc-contribution
bea953a5cf 2022-09-06 Angular Robot build: update dependency typescript to ~4.8.0 (#47293) upstream/main, upstream/HEAD, origin/main, origin/HEAD, main
cf0c53aa1c 2022-08-22 JoostK fix(compiler): avoid errors for inputs with Object-builtin names (#47220) upstream/g3
4a53b7f36b 2021-10-31 JoostK refactor(compiler): show meaningful diagnostic when reporting a template error fails (#44001)
```

10. The `git rebase` command changes the commit log in your local computer so it is now different from the one in your online repo. To update your online repo, you must force your push of changes from your local computer using this command.

    ```
    git push --force-with-lease
    ```

    This action is also called a *force push* because it changes the commit log that was stored in your GitHub account. Normally, when you run `git push`, you add new commits to the online repo. When other people have forked a repo, a force push can have undesired effects for them. This force-push is to your forked repo, which should not be shared, so it should be OK.

11. After your force push updates the online repo, your pull request restarts the automated tests and notifies the reviewers of the update.

Next steps

Repeat these update steps as necessary to respond to all the feedback you receive.

After you address all the feedback and your pull request has been approved, it is merged into `angular/angular`. The changes in your pull request should appear in the documentation shortly afterwards.

After your pull request is merged into `angular/angular`, you can clean up your workspace.

Last reviewed on Wed Oct 12 2022

Finish up a documentation pull request

This topic describes how to keep your workspace tidy after your pull request is merged and closed.

Review the commit log of the upstream repo

This procedure confirms that your commit is now in the `main` branch of the `angular/angular` repo.

To review the commit log on `github.com` for your commit

In a web browser, open `https://github.com/angular/angular/commits/main`.

1. Review the commit list.
    a. Find the entry with your GitHub username, commit message, and pull request number of your commit. The commit number might not match the commit from your working branch because of how commits are merged.
    b. If you see your commit listed, your commit has been merged into `angular/angular` and you can continue cleaning up your workspace.
    c. If you don't see your commit in the list, you might need to wait before you retry this step. Do not continue cleaning your workspace until you see your commit listed in or after the log entry that contains `origin/main`.
    d. If you see your commit listed above the log entry that contains `origin/main`, then you might need to update your clone of the `angular/angular` repo again.

Update your fork from the upstream repo

After you see that the commit from your pull request has been merged into the upstream `angular/angular` repo, update your fork.

This procedure updates your clone of `personal/angular` on your local computer and then, the repo in the cloud.

To update your fork with the upstream repo

Perform these steps from a command-line tool on your local computer.

1. From your workspace directory, run this command to navigate to your working directory. Remember to replace `personal` with your GitHub username.

   ```
   cd personal/angular
   ```

2. Run this command to check out the `main` branch.

   ```
   git checkout main
   ```

3. Run this command to update the `main` branch in the `working` directory on your local computer from the upstream `angular/angular` repo.

   ```
   git fetch upstream
   git merge upstream/main
   ```

4. Run this command to update your `personal/angular` repo on `github.com` with the latest from the upstream `angular/angular` repo.

   ```
   git push
   ```

5. Run this command to review the commit log of your fork. The `main` branch on your local computer and your origin repo on `github.com` are now in sync with the upstream `angular/angular` repo. Run this command to list the recent commits.

```
git log --pretty=format:"%h %as %an %Cblue%s
%Cgreen%D"
```

6. In the output of the previous `git log` command, find the entry with your GitHub username, commit message, and pull request number of your commit. The commit number might not match the commit from your working branch because of how commits are merged.
   You should find the commit from your pull request in or near the log entry that contains `upstream/main`.

If you find the commit from your pull request in the correct place, you can continue to delete your working branch.

Delete the working branch

After you confirm that your pull request is merged into `angular/angular` and appears in the `main` branch of your fork, you can delete the `working` branch.

Because your working branch was merged into the `main` branch of your fork, and the pull request has been closed, you no longer need the `working` branch. It might be tempting to keep it around, just in case, but it is probably not necessary. If you keep all your old working branches, your repository can collect unnecessary clutter.

To delete your working branch

1. From your workspace directory, run this command to navigate to your working directory. Remember to replace `personal` with your GitHub username.

   ```
   cd personal/angular
   ```

2. Run this command to check out the `main` branch.

   ```
   git checkout main
   ```

3. Run this command to delete the working branch used in the pull request from your local computer. Replace `working-branch-name` with the name of your working branch.

   ```
   git branch -d working-branch-name
   ```

4. Run this command to delete the working branch from your `personal/angular` repo on `github.com`. Replace `working-branch-name` with the name of your working branch.

   ```
   git push -d origin working-branch-name
   ```

Next step

After you delete the working branch for your last issue, you're ready to select another issue to resolve.

Last reviewed on Wed Oct 12 2022