# Contributing

## Submitting issues

Any bug-related issue should come with a reproducible test-case, use the Plunker links in the docs to easily create one.

## Important notes

Please don't edit files in the `dist` subdirectory as they are generated via gulp. You'll find source code in the `src` subdirectory!

### Code style

Regarding code style like indentation and whitespace, \*\*follow the conventions you see used in the source already.\*\*

## Modifying the code

First, ensure that you have the latest [Node.js](http://nodejs.org/) and [npm](http://npmjs.org/) installed.

Test that gulp's CLI and Bower are installed by running `gulp --version` and `bower --version`. If the commands aren't found, run `npm install -g gulpjs/gulp#4.0 bower`. For more information about installing the tools, see the [getting started with gulp guide](https://github.com/gulpjs/gulp/blob/master/docs/getting-started.md) or [bower.io](http://bower.io/) respectively.

> Currently, AngularStrap uses gulp 4.0 which is not yet available on npm repository, so you will need to install it from github.

1. Fork and clone the repo.

1. Run `npm install` to install all build dependencies (including gulp).

1. Run `bower install` to install the front-end dependencies.

1. Run `gulp test` to test this project.

Assuming that you don't see any red, you're ready to go. Just be sure to run `gulp test` after making any changes, to ensure that nothing is broken.

## Submitting a Pull Request

Before you submit your pull request consider the following guidelines:

\* Search [GitHub](https://github.com/angular/angular.js/pulls) for an open or closed Pull Request

that relates to your submission. You don't want to duplicate effort.

\* Make your changes in a new git branch

```shell

git checkout -b my-fix-branch master

```

\* Create your patch, including appropriate test cases.

\* Follow our [Coding Rules](#coding-rules)

\* Commit your changes and create a descriptive commit message (the

commit message is used to generate release notes, please check out our

[commit message conventions](#commit-message-format) and our commit message presubmit hook

`validate-commit-msg.js`):

```shell

git commit -a

```

\* Build your changes locally to ensure all the tests pass

```shell

gulp test

```

\* Push your branch to Github:

```shell

git push origin my-fix-branch

```

\* In Github, send a pull request to `angular-strap:master`.

\* If we suggest changes then you can modify your branch, rebase and force a new push to your GitHub

repository to update the Pull Request:

```shell

git rebase master -i

git push -f

```

That's it! Thank you for your contribution!

When the patch is reviewed and merged, you can safely delete your branch and pull the changes

from the main (upstream) repository:

\* Delete the remote branch on Github:

```shell

git push origin --delete my-fix-branch

```

\* Check out the master branch:

```shell

git checkout master -f

```

\* Delete the local branch:

```shell

git branch -D my-fix-branch

```

\* Update your master with the latest upstream version:

```shell

git pull --ff upstream master

```

## Git Commit Guidelines

We have very precise rules over how our git commit messages can be formatted. This leads to \*\*more

readable messages\*\* that are easy to follow when looking through the \*\*project history\*\*. But also,

we use the git commit messages to \*\*generate the AngularJS change log\*\*.

### Commit Message Format

Each commit message consists of a \*\*header\*\*, a \*\*body\*\* and a \*\*footer\*\*. The header has a special

format that includes a \*\*type\*\*, a \*\*scope\*\* and a \*\*subject\*\*:

```

<type>(<scope>): <subject>

<BLANK LINE>

<body>

<BLANK LINE>

<footer>

```

Any line of the commit message cannot be longer 100 characters! This allows the message to be easier

to read on github as well as in various git tools.

### Type

Must be one of the following:

\* \*\*feat\*\*: A new feature

\* \*\*fix\*\*: A bug fix

\* \*\*docs\*\*: Documentation only changes

\* \*\*style\*\*: Changes that do not affect the meaning of the code (white-space, formatting, missing

semi-colons, etc)

\* \*\*refactor\*\*: A code change that neither fixes a bug or adds a feature

\* \*\*perf\*\*: A code change that improves performance

\* \*\*test\*\*: Adding missing tests

\* \*\*chore\*\*: Changes to the build process or auxiliary tools and libraries such as documentation

generation

### Scope

The scope could be anything specifying place of the commit change. For example `$location`,

`$browser`, `$compile`, `$rootScope`, `ngHref`, `ngClick`, `ngView`, etc...

### Subject

The subject contains succinct description of the change:

\* use the imperative, present tense: "change" not "changed" nor "changes"

\* don't capitalize first letter

\* no dot (.) at the end

###Body

Just as in the \*\*subject\*\*, use the imperative, present tense: "change" not "changed" nor "changes"

The body should include the motivation for the change and contrast this with previous behavior.

###Footer

The footer should contain any information about \*\*Breaking Changes\*\* and is also the place to

reference GitHub issues that this commit \*\*Closes\*\*.