# Contribution Guide

If you have any comment or advice, please report your [issue](https://github.com/eggjs/egg/issues),

or make any change as you wish and submit a [PR](https://github.com/eggjs/egg/pulls).

## Reporting New Issues

- Please specify what kind of issue it is.

- Before you report an issue, please search for related issues. Make sure you are not going to open a duplicate issue.

- Explain your purpose clearly in tags(see \*\*Useful Tags\*\*), title, or content.

Egg group members will confirm the purpose of the issue, replace more accurate tags for it, identify related milestone, and assign developers working on it.

Tags can be divided into two groups, `type` and `scope`.

- type: What kind of issue, e.g. `feature`, `bug`, `documentation`, `performance`, `support` ...

- scope: What did you modified. Which files are modified, e.g. `core: xx`, `plugin: xx`, `deps: xx`

### Useful Tags

- `support`: the issue asks helps from developers of our group. If you need helps to locate and handle problems or have any idea to improve Egg, mark it as `support`.

- `bug`: if you find a problem which possiblly could be a bug, please tag it as `bug`. Then our group members will review that issue. If it is confirmed as a bug by our group member, this issue will be tagged as `confirmed`.

- A confirmed bug will be resolved prior.

- If the bug has negative impact on running online application, it will be tagged as `critical`, which refers to top priority, and will be fixed ASAP!

- A bug will be fixed from lowest necessary version, e.g. A bug needs to be fixed from 0.9.x, then this issue will be tagged as `0.9`, `0.10`, `1.0`, `1.1`, referring that the bug is required to be fixed in those versions.

- `core: xx`: the issue is related to core, e.g. `core: loader` refers that the issue is related with `loader` config.

- `plugin: xx`: the issue is related to plugins. e.g. `plugin: session` refers that the issue is related to `session` plugin.

- `deps: xx`: the issue is related to `dependencies`, e.g. `deps:egg-cors` refers that the issue is related to `egg-cors`

- `chore: documentation`: the issue is about documentation. Need to modify documentation.

## Documentation

All features must be submitted along with documentations. The documentations should satify several requirements.

- Documentations must clarify one or more aspects of the feature, depending on the nature of feature: what it is, why it happens and how it works.

- It's better to include a series of procedues to explain how to fix the problem. You are also encourgaed to provide \*\*simple, but self-explanatory\*\* demo.

All demos should be compiled at [eggjs/examples](https://github.com/eggjs/examples) repository.

- Please provide essential urls, such as application process, terminology explainations and references.

## Submitting Code

### Pull Request Guide

If you are developer of egg repo and you are willing to contribute, feel free to create a new branch, finish your modification and submit a PR. Egg group will review your work and merge it to master branch.

```bash

# Create a new branch for development. The name of branch should be semantic, avoiding words like 'update' or 'tmp'. We suggest to use feature/xxx, if the modification is about to implement a new feature.

$ git checkout -b branch-name

# Run the test after you finish your modification. Add new test cases or change old ones if you feel necessary

$ npm test

# If your modification pass the tests, congradulations it's time to push your work back to us. Notice that the commit message should be wirtten in the following format.

$ git add . # git add -u to delete files

$ git commit -m "fix(role): role.use must xxx"

$ git push origin branch-name

```

Then you can create a Pull Request at [egg](https://github.com/eggjs/egg/pulls)

No one can garantee how much will be remembered about certain PR after some time. To make sure we can easily recap what happened previously, please provide the following information in your PR.

1. Need: What function you want to achieve (Generally, please point out which issue is related).

2. Updating Reason: Different with issue. Briefly describe your reason and logic about why you need to make such modification.

3. Related Testing: Briefly descirbe what part of testing is relevant to your modification.

4. User Tips: Notice for Egg users. You can skip this part, if the PR is not about update in API or potential compatibility problem.

### Style Guide

Eslint can help to identify styling issues that may exist in your code. Your code is required to pass the test from eslint. Run the test locally by `$ npm run lint`.

### Commit Message Format

You are encouraged to use [angular commit-message-format](https://github.com/angular/angular.js/blob/master/DEVELOPERS.md#-git-commit-guidelines) to write commit message. In this way, we could have a more trackable history and an automatically generated changelog.

```xml

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

<BLANK LINE>

<body>

<BLANK LINE>

<footer>

```

?1?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 nor 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

- deps: Updates about dependencies

?2?scope

The scope could be anything specifying place of the commit change. For example $location, $browser, $compile, $rootScope, ngHref, ngClick, ngView, etc...

?3?subject

Use succinct words to describe what did you do in the commit change.

?4?body

Feel free to add more content in the body, if you think subject is not self-explanatory enough, such as what it is the purpose or reasone of you commit.

?5?footer

- \*\*If the commit is a Breaking Change, please note it clearly in this part.\*\*

- related issues, like `Closes #1, Closes #2, #3`

- If there is a change about an old feaure or a new feature, please associate `doc` and `egg-core`, like `eggjs/egg-core#123`

e.g.

```

fix($compile): [BREAKING\_CHANGE] couple of unit tests for IE9

Older IEs serialize html uppercased, but IE9 does not...

Would be better to expect case insensitive, unfortunately jasmine does

not allow to user regexps for throw expectations.

Document change on eggjs/egg#123

Closes #392

BREAKING CHANGE:

Breaks foo.bar api, foo.baz should be used instead

```

Look at [these files](https://docs.google.com/document/d/1QrDFcIiPjSLDn3EL15IJygNPiHORgU1\_OOAqWjiDU5Y/edit) for more details.

### Principles of English Translations

We follow the normal principles of English articles when translating, however, due to there're some special principles of titles, we should follow these rules:

- For nouns, verbs, pronouns, adjectives and adverbs, capitalize the first character. For prepsitions, articles, conjections, interjections and auxiliary words, the first character should be in lowercase. \*\*the character of the first word and the last word for title should be capitalized, regardless of what it is\*\*.

- For proper nouns such as the direct reference of a variable or the name of a plugin, we must use backtick (underneath the 'Esc') to surround them and keep what they are in origin.

- For prepsitions more than 5 characters, their first characters should be also capitalized, otherwise not.

- For some very important titles or some fixed proper nouns such as methods of Http: POST,GET,PUT,DELETE, every charater can be capitalized (USE WITH CAUTION).

- If the article belongs to the form of O-V (Object-Verb) such as "Config Management", we'd better translate it as "Management Configuration", or "Managing Configuration" in the form of "gerund+noun".

- If your title is taken as a sentence, write in 'Sentence Case' (e.g: In FAQ, each title is actually an English sentence).

For more info, please refer [English Title Case].

## Release Management

Egg uses semantic versioning in release process based on [semver].

### Branch Strategy

`master` branch is the latest stable version. `next` branch is the next stable version working in progress.

- All new features will be added into `master` or `next` branch as well as all bug-fix except security issues. In such way, we can motivate developers to update to the latest stable version.

- If any API is discarded, it should be noted with `deprecate` in current stable version. The old version of API should be compatiable until the release of next stable version.

- `master` branch doesn't have publish tag. High-level framework can work with stable versions defined by semantic versioning.

- `next` branch is labelled with `next` tag, high-level framework can use `egg@next` to test the in-progress version.

- The LTS versions of Egg determined by Milestone. If a version is listed in Milestone, then it is a LTS version. We will patch it if there is any problem with it.

### Release Strategy

In the release of every stable version, there will be a PM who has the following responsibilities in different stages of the release.

#### Preparation

- Set up milestone. Confirm that request is related to milestone. Assign and update issues, like [1.x milestone].

- Create a `next` branch from `master` branch and tag it as `next`.

#### Before Release

- Confirm that performance test is passed and all issues in current Milestone are either closed or can be delayed to later versions.

- Open a new [Release Proposal MR], and write `History` as [node CHANGELOG]. Don't forget to correct content in documentation which is related to the releasing version. Commits can be generated automatically.

```bash

$ npm run commits

```

- Nominate PM for next stable version.

#### During Release

- Back up the stable version (master) onto the branch named after the current major (e.g: `1.x`), and set the tag to `release-{v}.x` (v is the current version like `release-1.x`).

- Push the `next` branch to `master`, make it to the last stable one and remove `next` tag, change the contents corresponding to the branch in README.

- Publish the latest stable version to [npm], and notify the previous framework to be upgraded.

- Before doing `npm publish`, please read [How to deploy an npm package].

All tags mentioned above means the tags of npm in `package.json`.

```json

"publishConfig": {

"tag": "next"

}

```

[semver]: https://semver.org/

[Release Proposal MR]: https://github.com/nodejs/node/pull/4181

[node CHANGELOG]: https://github.com/nodejs/node/blob/master/CHANGELOG.md

[1.x milestone]: https://github.com/eggjs/egg/milestone/1

[npm]: http://npmjs.com/

[How to deploy an npm package]: https://fengmk2.com/blog/2016/how-i-publish-a-npm-package

[English Title Case]: https://headlinecapitalization.com/