# Contributing to dotNetty

dotNetty is a large project and contributions are more than welcome, so thank you for wanting to contribute to dotNetty!

---

### Checklist before creating a Pull Request

Submit only relevant commits. We don't mind many commits in a pull request, but they must be relevant as explained below.

- \_\_Use a feature branch\_\_ The pull request should be created from a feature branch, and not from \_dev\_. See below for why.

- \_\_No merge-commits\_\_

If you have commits that looks like this \_"Merge branch 'my-branch' into dev"\_ or \_"Merge branch 'dev' of https://github.com/Azure/DotNetty into dev"\_ you're probaly using merge instead of [rebase](https://help.github.com/articles/about-git-rebase) locally. See below on \_Handling updates from upstream\_.

- \_\_Squash commits\_\_ Often we create temporary commits like \_"Started implementing feature x"\_ and then \_"Did a bit more on feature x"\_. Squash these commits together using [interactive rebase](https://help.github.com/articles/about-git-rebase). Also see [Squashing commits with rebase](http://gitready.com/advanced/2009/02/10/squashing-commits-with-rebase.html).

- \_\_Descriptive commit messages\_\_ If a commit's message isn't descriptive, change it using [interactive rebase](https://help.github.com/articles/about-git-rebase). Refer to issues using `#issue`. Example of a bad message ~~"Small cleanup"~~. Example of good message: \_"Removed Security.Claims header from FSM, which broke Mono build per #62"\_. Don't be afraid to write long messages, if needed. Try to explain \_why\_ you've done the changes. The Erlang repo has some info on [writing good commit messages](https://github.com/erlang/otp/wiki/Writing-good-commit-messages).

- \_\_No one-commit-to-rule-them-all\_\_ Large commits that changes too many things at the same time are very hard to review. Split large commits into smaller. See this [StackOverflow question](http://stackoverflow.com/questions/6217156/break-a-previous-commit-into-multiple-commits) for information on how to do this.

- \_\_Tests\_\_ Add relevant tests and make sure all existing ones still passes. Tests can be run using the command

- \_\_No Warnings\_\_ Make sure your code do not produce any build warnings.

After reviewing a Pull request, we might ask you to fix some commits. After you've done that you need to force push to update your branch in your local fork.

#### Title and Description for the Pull Request ####

Give the PR a descriptive title and in the description field describe what you have done in general terms and why. This will help the reviewers greatly, and provide a history for the future.

Especially if you modify something existing, be very clear! Have you changed any algorithms, or did you just intend to reorder the code? Justify why the changes are needed.

---

### Getting started

Make sure you have a [GitHub](https://github.com/) account.

- Fork, clone, add upstream to the dotNetty repository. See [Fork a repo](https://help.github.com/articles/fork-a-repo) for more detailed instructions or follow the instructions below.

- Fork by clicking \_Fork\_ on https://github.com/Azure/DotNetty

- Clone your fork locally.

```

git clone https://github.com/YOUR-USERNAME/DotNetty

```

- Add an upstream remote.

```

git remote add upstream https://github.com/Azure/DotNetty

```

You now have two remotes: \_upstream\_ points to https://github.com/Azure/DotNetty, and \_origin\_ points to your fork on GitHub.

- Make changes. See below.

Unsure where to start? Issues marked with [\_up for grabs\_](https://github.com/Azure/DotNetty/labels/up%20for%20grabs) are things we want help with.

See also: [Contributing to Open Source on GitHub](https://guides.github.com/activities/contributing-to-open-source/)

New to Git? See https://help.github.com/articles/what-are-other-good-resources-for-learning-git-and-github

### Making changes

\_\_Never\_\_ work directly on \_dev\_ or \_master\_ and you should never send a pull request from master - always from a feature branch created by you.

- Pick an [issue](https://github.com/Azure/DotNetty/issues). If no issue exists (search first) create one.

- Get any changes from \_upstream\_.

```

git checkout dev

git fetch upstream

git merge --ff-only upstream/dev

git push origin dev #(optional) this makes sure dev in your own fork on GitHub is up to date

```

See https://help.github.com/articles/fetching-a-remote for more info

- Create a new feature branch. It's important that you do your work on your own branch and that it's created off of \_dev\_. Tip: Give it a descriptive name and include the issue number, e.g. `implement-lengthframeencoder-323` or `295-implement-recvbuffer`, so that others can see what is being worked on.

```

git checkout -b my-new-branch-123

```

- Work on your feature. Commit.

- Rebase often, see below.

- Make sure you adhere to \_Checklist before creating a Pull Request\_ described above.

- Push the branch to your fork on GitHub

```

git push origin my-new-branch-123

```

- Send a Pull Request, see https://help.github.com/articles/using-pull-requests to the \_dev\_ branch.

See also: [Understanding the GitHub Flow](https://guides.github.com/introduction/flow/) (we're using `dev` as our master branch)

### Handling updates from upstream

While you're working away in your branch it's quite possible that your upstream \_dev\_ may be updated. If this happens you should:

- [Stash](http://git-scm.com/book/en/Git-Tools-Stashing) any un-committed changes you need to

```

git stash

```

- Update your local \_dev\_ by fetching from \_upstream\_

```

git checkout dev

git fetch upstream

git merge --ff-only upstream/dev

```

- Rebase your feature branch on \_dev\_. See [Git Branching - Rebasing](http://git-scm.com/book/en/Git-Branching-Rebasing) for more info on rebasing

```

git checkout my-new-branch-123

git rebase dev

git push origin dev #(optional) this makes sure dev in your own fork on GitHub is up to date

```

This ensures that your history is "clean" i.e. you have one branch off from \_dev\_ followed by your changes in a straight line. Failing to do this ends up with several "messy" merges in your history, which we don't want. This is the reason why you should always work in a branch and you should never be working in, or sending pull requests from \_dev\_.

If you're working on a long running feature then you may want to do this quite often, rather than run the risk of potential merge issues further down the line.

### Making changes to a Pull request

If you realize you've missed something after submitting a Pull request, just commit to your local branch and push the branch just like you did the first time. This commit will automatically be included in the Pull request.

If we ask you to change already published commits using interactive rebase (like squashing or splitting commits or rewriting commit messages) you need to force push using `-f`:

```

git push -f origin my-new-branch-123

```

### All my commits are on dev. How do I get them to a new branch? ###

If all commits are on \_dev\_ you need to move them to a new feature branch.

You can rebase your local \_dev\_ on \_upstream/dev\_ (to remove any merge commits), rename it, and recreate \_dev\_

```

git checkout dev

git rebase upstream/dev

git branch -m my-new-branch-123

git branch dev upstream/dev

```

Or you can create a new branch off of \_dev\_ and then cherry pick the commits

```

git checkout -b my-new-branch-123 upstream/dev

git cherry-pick rev #rev is the revisions you want to pick

git cherry-pick rev #repeat until you have picked all commits

git branch -m dev old-dev #rename dev

git branch dev upstream/dev #create a new dev

```

## Code guidelines

See our [C# Coding Style](https://github.com/Azure/DotNetty/wiki/C%23-Coding-Style) for more information on following the project's conventions.

---

Props to [Akka.NET](http://getakka.net/) and [NancyFX](https://github.com/NancyFx/Nancy) from which we've "borrowed" this text.