# Developing locally

Since Kohana maintains many concurrent versions at once, there is no single `master` branch. All versions have branches named with a prefix of its version:

- 3.2/master

- 3.2/develop

- 3.3/master

- 3.3/develop

and so on. All development of versions happens in the develop branch of that version. Before a release, new features are added here. After a major release is actually released, only bugfixes can happen here. New features and API changes must happen in the develop branch of the next version.

## Branch name meanings

- \*\*3.3/master\*\* - master branches are for releases. Only release merge commits can be applied to this branch. You should never make a non-merge commit to this branch, and all merge commits should come from the release branch or hotfix branch (detailed below). This branch lasts forever.

- \*\*3.3/hotfix/\*\*\* - hotfix branches are for emergency maintenance after a release. If an important security or other kind of important issue is discovered after a release, it should be done here, and merged to master. This branch should be created from master and merged back into master and develop when complete. This branch is deleted after it's done.

- \*\*3.3/develop\*\* - If a version is not released, this branch is for merging features into. If the version is released, this branch is for applying bugfix commits to. This branch lasts forever.

- \*\*3.3/release/\*\*\* - release branches are for maintenance work before a release. This branch should be branched from the develop branch only. Change the version number/code name here, and apply any other maintenance items needed before actually releasing. Merges from master should only come from this branch. It should be merged to develop when it's complete as well. This branch is deleted after it's done.

- \*\*3.3/feature/\*\*\* - Details on these branches are outlined below. This branch is deleted after it's done.

If an bug/issue applies to multiple versions of Kohana, it is first fixed in the lowest supported version it applies to, then merged to each higher branch it applies to. Each merge should only happen one version up. 3.1 should merge to 3.2, and 3.2 should merge to 3.3. 3.1 should not merge directly to 3.3.

To work on a specific release branch you need to check it out then check out the appropriate system branch.

Release branch names follow the same convention in both kohana/kohana and kohana/core.

To work on 3.3.x you'd do the following:

> git clone git://github.com/kohana/kohana.git

# ....

> cd kohana

> git submodule update --init

# ....

> git checkout 3.3/develop

# Switched to branch '3.3/develop'

> git submodule foreach "git fetch && git checkout 3.3/develop"

# ...

It's important that you follow the last step, because unlike SVN, Git submodules point at a

specific commit rather than the tip of a branch. If you cd into the system folder after

a `git submodule update` and run `git status` you'll be told:

# Not currently on any branch.

nothing to commit (working directory clean)

\*\*\*

# Contributing to the project

All features and bugfixes must be fully tested and reference an issue in [GitHub](https://github.com/kohana/kohana/issues), \*\*there are absolutely no exceptions\*\*.

It's highly recommended that you write/run unit tests during development as it can help you pick up on issues early on. See the Unit Testing section below.

## Creating new features

New features or API breaking modifications should be developed in separate branches so as to isolate them

until they're stable.

\*\*Features without tests written will be rejected! There are NO exceptions.\*\*

The naming convention for feature branches is:

{version}/feature/{issue number}-{short hyphenated description}

// e.g.

3.2/feature/4045-rewriting-config-system

When a new feature is complete and fully tested it can be merged into its respective release branch using

`git pull --no-ff`. The `--no-ff` switch is important as it tells Git to always create a commit

detailing what branch you're merging from. This makes it a lot easier to analyse a feature's history.

Here's a quick example:

> git status

# On branch 3.2/feature/4045-rewriting-everything

> git checkout 3.1/develop

# Switched to branch '3.1/develop'

> git merge --no-ff 3.2/feature/4045-rewriting-everything

\*\*If a change you make intentionally breaks the API then please correct the relevant tests before pushing!\*\*

## Bug fixing

If you're making a bugfix then before you start create a unit test which reproduces the bug,

using the `@ticket` notation in the test to reference the bug's issue number

(e.g. `@ticket 4045` for issue #4045).

If you run the unit tests then the one you've just made should fail.

Once you've written the bugfix, run the tests again before you commit to make sure that the

fix actually works, then commit both the fix and the test.

\*\*Bug fixes without tests written will be rejected! There are NO exceptions.\*\*

There is no need to create separate branches for bugfixes, creating them in the main develop

branch is perfectly acceptable.

## Tagging releases

Tag names should be prefixed with a `v`, this helps to separate tag references from branch references in Git.

For example, if you were creating a tag for the `3.1.0` release the tag name would be `v3.1.0`

# Merging changes from remote repositories

Now that you have a remote repository, you can pull changes in the remote "kohana" repository

into your local repository:

> git pull kohana 3.1/master

\*\*Note:\*\* Before you pull changes you should make sure that any modifications you've made locally

have been committed.

Sometimes a commit you've made locally will conflict with one made in the remote "kohana" repo.

There are a couple of scenarios where this might happen:

## The conflict is due to a few unrelated commits and you want to keep changes made in both commits

You'll need to manually modify the files to resolve the conflict, see the "Resolving a merge"

section [in the Git SCM book](http://book.git-scm.com/3\_basic\_branching\_and\_merging.html) for more info

## You've fixed something locally which someone else has already done in the remote repo

The simplest way to fix this is to remove all the changes that you've made locally.

You can do this using

> git reset --hard kohana

## You've fixed something locally which someone else has already fixed but you also have separate commits you'd like to keep

If this is the case then you'll want to use a tool called rebase. First of all we need to

get rid of the conflicts created due to the merge:

> git reset --hard HEAD

Then find the hash of the offending local commit and run:

> git rebase -i {offending commit hash}

i.e.

> git rebase -i 57d0b28

A text editor will open with a list of commits. Delete the line containing the offending commit

before saving the file & closing your editor.

Git will remove the commit and you can then pull/merge the remote changes.

# Unit Testing

Kohana currently uses PHPUnit for unit testing. This is installed with composer.

## How to run the tests

\* Install [Phing](http://phing.info)

\* Make sure you have the [unittest](http://github.com/kohana/unittest) module enabled.

\* Install [Composer](http://getcomposer.org)

\* Run `php composer.phar install` from the root of this repository

\* Finally, run `phing test`

This will run the unit tests for core and all the modules and tell you if anything failed. If you haven't changed anything and you get failures, please create a new issue on [the tracker](http://dev.kohanaframework.org) and paste the output (including the error) in the issue.