# Guidelines for Getting Help with UserFrosting

\*\*Before\*\* you open a new issue or ask a question in chat, you \*\*must\*\* read these guidelines. If it is evident from your issue that you failed to research your question properly, your issue may be closed without being answered.

## Troubleshooting

There are a few common stumbling blocks that new users face when setting up UserFrosting for the first time. If you are new to the current version of UserFrosting, please first look at the [basic requirements and installation instructions](https://learn.userfrosting.com/basics/requirements/basic-stack).

If you don't find what you're looking for in the troubleshooting page, then please check the [wiki](https://github.com/userfrosting/UserFrosting/wiki) and [existing issues](https://github.com/alexweissman/UserFrosting/issues?utf8=%E2%9C%93&q=is%3Aissue), both opened and closed. Your question may have already been asked and answered before!

You can also search for help on Stack Overflow or in our [Forums](https://forums.userfrosting.com/). In addition to the tags for the components that UF builds upon, such as [Slim](http://stackoverflow.com/questions/tagged/slim), [Twig](http://stackoverflow.com/questions/tagged/twig), [Eloquent](http://stackoverflow.com/questions/tagged/eloquent), [jQuery Validate](http://stackoverflow.com/questions/tagged/jquery-validate), [Select2](http://stackoverflow.com/questions/tagged/jquery-select2), there is now a [UserFrosting tag](http://stackoverflow.com/questions/tagged/userfrosting) as well.

There are also tags for the utilities upon which UserFrosting depends, such as [Composer](http://stackoverflow.com/questions/tagged/composer-php) and [Git](http://stackoverflow.com/questions/tagged/git).

## Asking for Help

In general, the Github issue tracker should only be used for bug reports and feature requests. If you're just having trouble getting something to work, you should ask on Stack Overflow or in our [Forums](https://forums.userfrosting.com/) instead. Tag your question with the `userfrosting` tag, and optionally with any tags specific to the relevant underlying technologies, such as `slim`, `twig`, `eloquent`, `composer`, etc. You should also mention the version of UserFrosting that you are using.

After posting a question on Stack Overflow or in our [Forums](https://forums.userfrosting.com/), please [link to it in chat](https://chat.userfrosting.com). This will ensure that more people see it, and provide a place where we can discuss and help clarify your question.

On Github, Chat, and Stack Overflow, please keep in mind the following:

1. Remember that courtesy and proper grammar go a long way. Please take the time to craft a \*\*precise, polite issue\*\*. We will do our best to help, but remember that this is an open-source project - none of us are getting paid a salary to develop this project, or act as your personal support hotline :wink:

2. Report any errors in detail. Vague issues like "it doesn't work when I do this" are not helpful. Show that you have put some effort into identifying the cause of the error.

3. There are three main places where you may find error messages:

- Backend (PHP-related) fatal errors: in your PHP error log. This is usually a file called `php\_error\_log` or something like that. In XAMPP, the default location of this file is `XAMPP/xamppfiles/logs/`. For other web hosting platforms, please consult the documentation or do a quick Google search (i.e. "where is the php error log in \_\_\_\_\_"). Some web hosts may provide a special interface for accessing the php error log, through ssh, cpanel, etc. Please ask them directly for help with this.

- Non-fatal PHP errors will be logged in your UserFrosting error log. Look for your `app/logs/errors.log` file.

- Frontend (Javascript-related) errors: in your browser's Javascript console. See [this guide](https://learn.userfrosting.com/background/client-side) to using your browser console.

You should also try testing your code in a local development environment, to separate \*\*code-related\*\* issues from \*\*server\*\* issues. In general, we recommend that you install a local development server on your computer, rather than [testing your code directly on the production server](https://pbs.twimg.com/media/BxfENwpIYAAcHqQ.png). This means you can test your code directly on your own computer, making development faster and without the risk of exposing sensitive information to the public. We recommend installing [XAMPP](https://www.apachefriends.org) if you don't already have a local server set up.

## Contributing to the Codebase

We welcome your technical expertise! But first, please join us in [chat](https://chat.userfrosting.com) to discuss your proposed changes/fixes/enhancements before you get started. At least one member of our development team will usually be around.

Please also be sure to read our [style guidelines](../STYLE-GUIDE.md).

When it's time to integrate changes, our git flow more or less follows http://nvie.com/posts/a-successful-git-branching-model/.

### Branches

#### `master`

The current release or release candidate. Always numbered as `major.minor.revision`, possibly with an `-alpha`, `-beta` or `-RC` extension as well. Commits should \*\*never\*\* be send directly on this branch.

#### `hotfix`

Contains the next bug fix release, typically matching the next `revision` version. Any changes not introducing a breaking change can be committed to this branch. Always numbered as `major.minor.revision`.

When ready, changes should be merged into both \*\*master\*\* and \*\*develop\*\*.

#### `develop`

Contains breaking changes that will need to wait for the next version to be integrated. Typically matched the next `minor` version. Always numbered as `major.minor.x`.

When ready, changes should be merged into both \*\*master\*\* and \*\*hotfix\*\*.

#### `feature-\*`

New features that introduce some breaking changes or incomplete code should be committed in a separate `feature-{name}` branch.

When ready, the branch should be \*\*[squashed-merged](https://github.com/blog/2141-squash-your-commits)\*\* ([guide](https://stackoverflow.com/a/5309051/445757)) into `develop` (or `hotfix` if it doesn't introduce a breaking change).

### Releases

After every release, the `hotfix` branch (and possibly `develop`, for minor/major releases) should immediately be version-bumped. That way, new changes can be accumulated until the next release.

When a new version is created, the version number need to be changed in `app/define.php`. `CHANGELOG.md` should also be updated and the associated tag should be created on Github.

#### Alpha/beta/RC releases

During alpha/beta/RC, a release candidate always sits on the `master` branch. During the alpha/beta phase, major changes can still be integrated into `master` from `develop`. However, this should bump the revision number instead of the minor/major number. During RC, only \_hotfixes\_ can be merged into `master`.

## Working together

### Issues

Issues are used as a todo list. Each issue represent something that needs to be fixed, added or improved. Be sure to assign issues to yourself when working on something so everyones knows this issue is taken care of.

Issues are tagged to represent the feature or category it refers to. We also have some special tags to help organize issues. These includes:

- [`good first issue`](https://github.com/userfrosting/UserFrosting/labels/good%20first%20issue): If this is your first time contributing to UserFrosting, look for the `good first issue` tag. It's associated with easier issues anyone can tackle.

- [`up-for-grabs`](https://github.com/userfrosting/UserFrosting/labels/up-for-grabs): Theses issues have not yet been assigned to anybody. Look for theses when you want to start working on a new issue.

- [`needs discussion`](https://github.com/userfrosting/UserFrosting/labels/needs%20discussion) : This issue needs to be discussed with the dev team before being implemented as more information is required, questions remain or a higher level decision needs to be made.

- [`needs more info`](https://github.com/userfrosting/UserFrosting/labels/needs%20more%20info): More information is required from the author of the issue.

### Milestones

In order to keep a clear roadmap, milestones are used to track what is happening and what needs to be done. Milestones are used to classify problems by:

- Things that need to be done ASAP

- Things we are doing right now

- Things we will probably do soon

- Things we probably will not do soon

\*\*Things that need to be done ASAP\*\*: this is the highest priority and this milestone should always be empty. Issues related to important bug fixes should be set on this milestone immediately. The milestone always refers to the next version of \_revision\_, also known as the next bugfix version.

\*\*Things we are doing right now\*\*: this is the "main" milestone we are currently working on. Usually represents the next `minor` version, but may also represent the next major version when the focus is on the next major release.

\*\*Things well probably do soon\*\*: It's a "Next Tasks" milestone. These tasks will be addressed in the near future, but not close enough for the next version. Usually represents the second minor revision \*\*and\*\* the next major release.

\*\*Things we probably wont do soon\*\*: We refer to these issues and sometimes look through them, but they are easy to ignore and sometimes intentionally ignored. Represent issues without milestones that do not have a defined timeframe.

To maintain a clear history of progress on each milestone, milestones must be closed when completed and the corresponding version released. A new milestone must then be created for the next release. In addition, the milestone version must be updated when new versions are released.

## Learn documentation

The [Learn Documentation](https://learn.userfrosting.com) should always be updated along side code changes.

Changes to the [learn repository](https://github.com/userfrosting/learn) should follow the same logic as the main repository, ie. any changes applied to the `hotfix` branch should be documented in the learn `hotfix` branch. This also apply to `feature-\*` branches.

Additionally, the `learn` repository can have `dev-\*` for learn specific features and fixes.

## Building the API documentation

To build the API documentation, install [phpDocumentor](https://www.phpdoc.org) globally and then run from the UserFrosting root :

```

phpdoc

```

The resulting documentation will be available in `api/`.

## Automatically fixing coding style with PHP-CS-Fixer

[PHP-CS-Fixer](https://github.com/FriendsOfPHP/PHP-CS-Fixer) can be used to automatically fix PHP code styling. UserFrosting provides a project specific configuration file ([`.php\_cs`](.php\_cs)) with a set of rules reflecting our [style guidelines](../STYLE-GUIDE.md). This tool should be used before submitting any code change to assure the style guidelines are met. Every sprinkles will also be parsed by the fixer.

PHP-CS-Fixer is automatically loaded by Composer and can be used from the UserFrosting root directory :

```

app/vendor/bin/php-cs-fixer fix

```

## Useful tools

If you are using \*\*Atom\*\*, be sure to checkout theses useful packages :

- [Docblockr](https://atom.io/packages/docblockr) : Used to generate [documentation block](https://github.com/userfrosting/UserFrosting/blob/master/STYLE-GUIDE.md#documentation).

- [php-ide-serenata](https://atom.io/packages/php-ide-serenata) : Integrates [Serenata](https://gitlab.com/Serenata/Serenata) as PHP IDE, providing autocompletion, code navigation, refactoring, signature help, linting and annotations.