CONTRIBUTING GUIDELINES

Oh-My-Zsh is a community-driven project. Contribution is welcome, encouraged, and appreciated. It is also essential for the development of the project.

First, please take a moment to review our code of conduct.

These guidelines are an attempt at better addressing the huge amount of pending issues and pull requests. Please read them closely.

Foremost, be so kind as to search. This ensures any contribution you would make is not already covered.

Reporting Issues

You have a problem

Please be so kind as to search for any open issue already covering your problem.

If you find one, comment on it so we can know there are more people experiencing it.

If not, look at the Troubleshooting page for instructions on how to gather data to better debug your problem.

Then, you can go ahead and create an issue with as much detail as you can provide. It should include the data gathered as indicated above, along with:

1. How to reproduce the problem
2. What the correct behavior should be
3. What the actual behavior is

Please copy to anyone relevant (*eg* plugin maintainers) by mentioning their GitHub handle (starting with `@`) in your message.

We will do our very best to help you.

You have a suggestion

Please be so kind as to search for any open issue already covering your suggestion.

If you find one, comment on it so we can know there are more people supporting it.

If not, you can go ahead and create an issue. Please copy to anyone relevant (*eg* plugin maintainers) by mentioning their GitHub handle (starting with `@`) in your message.

Submitting Pull Requests

Getting started

You should be familiar with the basics of contributing on GitHub and have a fork properly set up.

You MUST always create PRs with *a dedicated branch* based on the latest upstream tree.

If you create your own PR, please make sure you do it right. Also be so kind as to reference any issue that would be solved in the PR description body, for instance "*Fixes #XXXX*" for issue number XXXX.

You have a solution

Please be so kind as to search for any open issue already covering your problem, and any pending/merged/rejected PR covering your solution.

If the solution is already reported, try it out and +1 the pull request if the solution works ok. On the other hand, if you think your solution is better, post it with a reference to the other one so we can have both solutions to compare.

If not, then go ahead and submit a PR. Please copy to anyone relevant (e.g. plugin maintainers) by mentioning their GitHub handle (starting with `@`) in your message.

You have an addition

Please do not send themes for now.

Please be so kind as to search for any pending, merged or rejected Pull Requests covering or related to what you want to add.

If you find one, try it out and work with the author on a common solution.

If not, then go ahead and submit a PR. Please copy to anyone relevant (*eg* plugin maintainers) by mentioning their GitHub handle (starting with `@`) in your message.

For any extensive change, *eg* a new plugin, you will have to find testers to +1 your PR.

Use the Search, Luke

May the Force (of past experiences) be with you

GitHub offers many search features to help you check whether a similar contribution to yours already exists. Please search before making any contribution, it avoids duplicates and eases maintenance. Trust me, that works 90% of the time.

You can also take a look at the FAQ to be sure your contribution has not already come up.

If all fails, your thing has probably not been reported yet, so you can go ahead and create an issue or submit a PR.

Commit Guidelines

Oh My Zsh uses the Conventional Commits specification. The automatic changelog tool uses these to automatically generate a changelog based on the commit messages. Here's a guide to writing a commit message to allow this:

Format

```
type(scope)!: subject
```

- `type` : the type of the commit is one of the following:
  - `feat` : new features.
  - `fix` : bug fixes.
  - `docs` : documentation changes.
  - `refactor` : refactor of a particular code section without introducing new features or bug fixes.
  - `style` : code style improvements.
  - `perf` : performance improvements.
  - `test` : changes to the test suite.
  - `ci` : changes to the CI system.
  - `build` : changes to the build system (we don't yet have one so this shouldn't apply).
  - `chore` : for other changes that don't match previous types. This doesn't appear in the changelog.
- `scope` : section of the codebase that the commit makes changes to. If it makes changes to many sections, or if no section in particular is modified, leave blank without the parentheses. Examples:
  - Commit that changes the `git` plugin:

  ```
  feat(git): add alias for `git commit`
  ```

  - Commit that changes many plugins:

  ```
  style: fix inline declaration of arrays
  ```

  For changes to plugins or themes, the scope should be the plugin or theme name:

  - ✅ `fix(agnoster): commit subject`
  - ❌ `fix(theme/agnoster): commit subject`
- `!` : this goes after the `scope` (or the `type` if scope is empty), to indicate that the commit introduces breaking changes.

  Optionally, you can specify a message that the changelog tool will display to the user to indicate what's changed and what they can do to deal with it. You can use multiple lines to type this message; the changelog parser will keep reading until the end of the commit message or until it finds an empty line.

  Example (made up):

  ```
  style(agnoster)!: change dirty git repo glyph

  BREAKING CHANGE: the glyph to indicate when a git repository is dirty has
  changed from a Powerline character to a standard UTF-8 emoji. You can
  change it back by setting `ZSH_THEME_DIRTY_GLYPH`.

  Fixes #420

  Co-authored-by: Username <email>
  ```
- `subject` : a brief description of the changes. This will be displayed in the changelog. If you need to specify other details you can use the commit body but it won't be visible.

Formatting tricks: the commit subject may contain:

- Links to related issues or PRs by writing `#issue`. This will be highlighted by the changelog tool:

  ```
  feat(archlinux): add support for aura AUR helper (#9467)
  ```

- Formatted inline code by using backticks: the text between backticks will also be highlighted by the changelog tool:

  ```
  feat(shell-proxy): enable unexported `DEFAULT_PROXY` setting (#9774)
  ```

Style

Try to keep the first commit line short. This is harder to do using this commit style but try to be concise and if you need more space, you can use the commit body. Try to make sure that the commit subject is clear and precise enough that users will know what change by just looking at the changelog.

Volunteer

Very nice!! :)

Please have a look at the Volunteer page for instructions on where to start and more.