> *This wiki is automatically published from [ohmyzsh/wiki](). To edit this page, go to [ohmyzsh/wiki](), make your changes and submit a Pull Request.*

Oh My Zsh is fully configurable to your needs with the help of the `$ZSH_CUSTOM` variable, whether you want to change its internals, plugins or the rich theme set – and all that **without forking**!

Initially `$ZSH_CUSTOM` points to Oh My Zsh's `custom` directory. Whatever you place inside of it will take precedence over the built-in configuration.

Overriding and adding plugins

Adding a new plugin

Let's say you created your own plugin `foobar` and want to add it to your configuration.

`~/.zshrc`

```
plugins=(git bundler foobar)
```

Then, create a `foobar` directory inside the `plugins` folder and an initialization script to launch your plugin. This script has to follow a naming convention, as all plugin files must have an ending of `.plugin.zsh`. Your file tree should look like this:

```
$ZSH_CUSTOM
└── plugins
    └── foobar
        └── foobar.plugin.zsh
```

Overriding an existing plugin

Also follow these steps if you want to override plugins that ship with your Oh My Zsh installation. To override a plugin with a custom version, put your custom version at `$ZSH_CUSTOM/plugins/<plugin_name>/`. For example, if it's the rvm plugin you want to override, create the directory `custom/plugins/rvm` and place a file called `rvm.plugin.zsh` inside of it.

This method will override the entire plugin: your custom plugin files will be loaded *instead* of the files from the original plugin.

Partially overriding an existing plugin

To partially customize a plugin by redefining individual functions or variables from it, use a "patch" plugin. Create a custom "patch" plugin with a different name that has definitions for just the items you wish to override, and load that after the base plugin by putting your patch plugin in `$plugins` *after* the base plugin it customizes. (This is necessary because user custom files are loaded before any plugins are loaded.) Make sure it doesn't have the same name as any other base plugin, either.

This may or may not work, depending on how the plugin is written: your redefinitions only take effect after the plugin has been loaded by `source`-ing its definition file. If any of the things you redefine are called or referenced during the plugin's initialization, they will not see your redefinitions.

You can combine customizations to multiple plugins in a single custom patch plugin that is loaded after all of the plugins you're customizing, as long as none of the plugins have load-time dependencies on definitions that you're modifying from earlier-loaded plugins.

Overriding and adding themes

Adding and customizing your own themes pretty much works the same as with plugins.

Themes are located in a `themes` folder and must end with `.zsh-theme`. The basename of the file is the name of the theme.

```
$ZSH_CUSTOM
└── themes
    └── my_awesome_theme.zsh-theme
```

Then edit your .zshrc to use that theme.

```
ZSH_THEME="my_awesome_theme"
```

Remember that customizations always take precedence over built-ins. If you happen to enjoy a particular theme that comes packaged with Oh My Zsh, but would like to change just a little detail inside of it – let's say you love the `agnoster` theme, it will be the easiest to copy the `agnoster.zsh-theme` file to your `custom/themes` directory and customize it.

If you don't change its filename, your `.zshrc` file can stay the same: `ZSH_THEME="agnoster"` will be perfect and still take your changes into account. You might also want to consider this before filing a new issue or pull request that just changes a trivial detail inside of a built-in theme.

Overriding internals

Oh My Zsh's internals are defined in its `lib` directory. To change them, just create a file inside the `custom` directory (its name doesn't matter, as long as it has a `.zsh` ending) and start customizing whatever you want. Unsatisfied with the way `git_prompt_info()` works? Write your own implementation!

`$ZSH_CUSTOM/my_patches.zsh`

```
function git_prompt_info() {
  # prove that you can do better
}
```

Such customization files will be loaded the last, after the built-in `lib/*.zsh` internals and plugins.

You can also fully override an existing `lib/*.zsh` file by providing a `$ZSH_CUSTOM/lib/<name>.zsh` file of the same name. It will be loaded instead of the corresponding base lib file. Note that files in this directory that do not have a corresponding base lib file of the same name will be ignored.

Using another customization directory

If you don't want to use the built-in `custom` directory itself, just change the path of `$ZSH_CUSTOM` inside your `.zshrc` to a directory of your own liking. Everything will be fine as long as you adhere to the conventional file hierarchy.

`~/.zshrc`

```
ZSH_CUSTOM=$HOME/my_customizations
```

File tree inside of your home directory:

```
$HOME
└── my_customizations
    ├── my_lib_patches.zsh
    ├── plugins
    │   └── my_plugin
    │       └── my_plugin.plugin.zsh
    └── themes
        └── my_awesome_theme.zsh-theme
```

Version control of customizations

By default, git is set to ignore the custom directory, so that Oh My Zsh's update process does not interfere with your customizations. If you want to use a version control system like git for your personal changes, just initialize your own repository inside the `custom` directory (`git init`), or point `$ZSH_CUSTOM` to another directory you have under version control.