



> On this page

SCSS style guide CONTRIBUTE

This style guide recommends best practices for SCSS to make styles easy to read, easy to maintain, and performant for the end-user.

Rules

Our CSS is a mixture of current and legacy approaches. That means sometimes it may be difficult to follow this guide to the letter; it means you are likely to run into exceptions, where following the guide is difficult to impossible without major effort. In those cases, you may work with your reviewers and maintainers to identify an approach that does not fit these rules. Try to limit these cases.

Utility Classes

In order to reduce the generation of more CSS as our site grows, prefer the use of utility classes over adding new CSS. In complex cases, CSS can be addressed by adding component classes.


Where are utility classes defined?

Prefer the use of [utility classes defined in GitLab UI](#).

An easy list of classes can also be [seen on Unpkg](#).

Classes in `utilities.scss` and `common.scss` are being deprecated. Classes in `common.scss` that use non-design-system values should be avoided. Use classes with conforming values instead.

Avoid [Bootstrap’s Utility Classes](#).

-  While migrating [Bootstrap’s Utility Classes](#) to the [GitLab UI](#) utility classes, note both the classes for margin and padding differ. The size scale used at GitLab differs from the scale used in the Bootstrap library. For a Bootstrap padding or margin utility, you may need to double the size of the applied utility to achieve the same visual result (such as `ml-1` becoming `gl-ml-2`).

Where should you put new utility classes?

If a class you need has not been added to GitLab UI, you get to add it! Follow the naming patterns documented in the [utility files](#) and refer to the [GitLab UI CSS documentation](#) for more details, especially about adding responsive and stateful rules.

If it is not possible to wait for a GitLab UI update (generally one day), add the class to `utilities.scss` following the same naming conventions documented in GitLab UI. A follow-up issue to backport the class to GitLab UI and delete it from GitLab should be opened.

When should you create component classes?

We recommend a “utility-first” approach.

1. Start with utility classes.
2. If composing utility classes into a component class removes code duplication and encapsulates a clear responsibility, do it.

This encourages an organic growth of component classes and prevents the creation of one-off non-reusable classes. Also, the kind of classes that emerge from “utility-first” tend to be design-centered (for example, `.button`, `.alert`, `.card`) rather than domain-centered (for example, `.security-report-widget`, `.commit-header-icon`).

Inspiration:

- <https://tailwindcss.com/docs/utility-first>

- <https://tailwindcss.com/docs/extracting-components> 


Utility mixins

In addition to utility classes GitLab UI provides utility mixins named after the utility classes.

For example a utility class `.gl-mt-3` will have a corresponding mixin `gl-mt-3`. Here's how it can be used in an SCSS file:

```
.my-class {  
  @include gl-mt-3;  
}
```

These mixins should be used to replace *magic values* in our code. For example a `margin-top: 8px` is a good candidate for the `@include gl-mt-3` mixin replacement.

Avoid using utility mixins for [pre-defined CSS keywords](#) . For example prefer `display: flex` over `@include gl-display-flex`. Utility mixins are particularly useful for encapsulating our design system but there is no need to encapsulate simple properties.

```
// Bad  
.my-class {  
  @include gl-display-flex;  
}  
  
// Good  
.my-class {  
  display: flex;  
}  
  
// Good  
.my-class {  
  @include gl-mt-3;  
}
```

Naming

Filenames should use `snake_case`.

CSS classes should use the `lowercase-hyphenated` format rather than `snake_case` or `camelCase`.

```
// Bad  
.class_name {  
  color: #fff;  
}  
  
// Bad  
.className {  
  color: #fff;  
}  
  
// Good  
.class-name {  
  color: #fff;  
}
```

Class names should be used instead of tag name selectors. Using tag name selectors is discouraged because they can affect unintended elements in the hierarchy.

```
// Bad
ul {
  color: #fff;
}

// Good
.class-name {
  color: #fff;
}

// Best
// prefer an existing utility class over adding existing styles
```

Class names are also preferable to IDs. Rules that use IDs are not-reusable, as there can only be one affected element on the page.

```
// Bad
#my-element {
  padding: 0;
}

// Good
.my-element {
  padding: 0;
}
```

Selectors with a `js-` Prefix

Do not use any selector prefixed with `js-` for styling purposes. These selectors are intended for use only with JavaScript to allow for removal or renaming without breaking styling.

Variables

Before adding a new variable for a color or a size, guarantee:

- There isn't an existing one.
- There isn't a similar one we can use instead.

Using `extend` at-rule

Usage of the `extend` at-rule is prohibited due to [memory leaks](#) and [the rule doesn't work as it should to](#) . Use mixins instead:

```
// Bad
.gl-pt-3 {
  padding-top: 12px;
}

.my-element {
  @extend .gl-pt-3;
}

// compiles to
.gl-pt-3, .my-element {
  padding-top: 12px;
}

// Good
@mixin gl-pt-3 {
  padding-top: 12px;
}

.my-element {
  @include gl-pt-3;
}

// compiles to
.my-element {
  padding-top: 12px;
}
```

Linting


We use [stylelint](#) to check for style guide conformity. It uses the ruleset in `.stylelintrc` and rules from [our SCSS configuration](#). `.stylelintrc` is located in the home directory of the project.

To check if any warnings are produced by your changes, run `yarn lint:stylelint` in the GitLab directory. Stylelint also runs in GitLab CI/CD to catch any warnings.

If the Rake task is throwing warnings you don't understand, SCSS Lint's documentation includes [a full list of their rules](#).

 **Help & feedback**





[Docs Repo](#) [About GitLab](#) [Terms](#) [Privacy Statement](#) [Contact](#)

View [page source](#) - Edit in [Web IDE](#) 