Theming Bootstrap

Customize Bootstrap 4 with our new built-in Sass variables for global style preferences for easy theming and component changes.

Introduction

In Bootstrap 3, theming was largely driven by variable overrides in LESS, custom CSS, and a separate theme stylesheet that we included in our `dist` files. With some effort, one could completely redesign the look of Bootstrap 3 without touching the core files. Bootstrap 4 provides a familiar, but slightly different approach.

Now, theming is accomplished by Sass variables, Sass maps, and custom CSS. There's no more dedicated theme stylesheet; instead, you can enable the built-in theme to add gradients, shadows, and more.

Sass

Utilize our source Sass files to take advantage of variables, maps, mixins, and more. In our build we've increased the Sass rounding precision to 6 (by default it's 5) to prevent issues with browser rounding.

File structure

Whenever possible, avoid modifying Bootstrap's core files. For Sass, that means creating your own stylesheet that imports Bootstrap so you can modify and extend it. Assuming you're using a package manager like npm, you'll have a file structure that looks like this:

Copy

```
your-project/
├── scss
│   └── custom.scss
└── node_modules/
    └── bootstrap
        ├── js
        └── scss
```

If you've downloaded our source files and aren't using a package manager, you'll want to manually setup something similar to that structure, keeping Bootstrap's source files separate from your own.

Copy

```
your-project/
├── scss
│   └── custom.scss
└── bootstrap/
    ├── js
    └── scss
```

Importing

In your `custom.scss`, you'll import Bootstrap's source Sass files. You have two options: include all of Bootstrap, or pick the parts you need. We encourage the latter, though be aware there are some requirements and dependencies across our components. You also will need to include some JavaScript for our plugins.

Copy

```
// Custom.scss
// Option A: Include all of Bootstrap

@import "../node_modules/bootstrap/scss/bootstrap";
```

Copy

```
// Custom.scss
// Option B: Include parts of Bootstrap

// Required
@import "../node_modules/bootstrap/scss/functions";
@import "../node_modules/bootstrap/scss/variables";
@import "../node_modules/bootstrap/scss/mixins";

// Optional
@import "../node_modules/bootstrap/scss/reboot";
@import "../node_modules/bootstrap/scss/type";
@import "../node_modules/bootstrap/scss/images";
@import "../node_modules/bootstrap/scss/code";
@import "../node_modules/bootstrap/scss/grid";
```

With that setup in place, you can begin to modify any of the Sass variables and maps in your `custom.scss`. You can also start to add parts of Bootstrap under the `// Optional` section as needed. We suggest using the full import stack from our `bootstrap.scss` file as your starting point.

Variable defaults

Every Sass variable in Bootstrap 4 includes the `!default` flag allowing you to override the variable's default value in your own Sass without modifying Bootstrap's source code. Copy and paste variables as needed, modify their values, and remove the `!default` flag. If a variable has already been assigned, then it won't be re-assigned by the default values in Bootstrap.

You will find the complete list of Bootstrap's variables in `scss/_variables.scss`. Some variables are set to `null`, these variables don't output the property unless they are overridden in your configuration.

Variable overrides within the same Sass file can come before or after the default variables. However, when overriding across Sass files, your overrides must come before you import Bootstrap's Sass files.

Here's an example that changes the `background-color` and `color` for the `<body>` when importing and compiling Bootstrap via npm:

Copy

```
// Your variable overrides
$body-bg: #000;
$body-color: #111;

// Bootstrap and its default variables
@import "../node_modules/bootstrap/scss/bootstrap";
```

Repeat as necessary for any variable in Bootstrap, including the global options below.

Maps and loops

Bootstrap 4 includes a handful of Sass maps, key value pairs that make it easier to generate families of related CSS. We use Sass maps for our colors, grid breakpoints, and more. Just like Sass variables, all Sass maps include the `!default` flag and can be overridden and extended.

Some of our Sass maps are merged into empty ones by default. This is done to allow easy expansion of a given Sass map, but comes at the cost of making *removing* items from a map slightly more difficult.

Modify map

To modify an existing color in our `$theme-colors` map, add the following to your custom Sass file:

Copy

```
$theme-colors: (
  "primary": #0074d9,
  "danger": #ff4136
);
```

Add to map

To add a new color to `$theme-colors`, add the new key and value:

Copy

```
$theme-colors: (
  "custom-color": #900
);
```

Remove from map

To remove colors from `$theme-colors`, or any other map, use `map-remove`. Be aware you must insert it between our requirements and options:

Copy

```
// Required
@import "../node_modules/bootstrap/scss/functions";
@import "../node_modules/bootstrap/scss/variables";
@import "../node_modules/bootstrap/scss/mixins";

$theme-colors: map-remove($theme-colors, "info", "light", "dark");

// Optional
@import "../node_modules/bootstrap/scss/root";
@import "../node_modules/bootstrap/scss/reboot";
@import "../node_modules/bootstrap/scss/type";
...
```

Required keys

Bootstrap assumes the presence of some specific keys within Sass maps as we used and extend these ourselves. As you customize the included maps, you may encounter errors where a specific Sass map's key is being used.

For example, we use the `primary`, `success`, and `danger` keys from `$theme-colors` for links, buttons, and form states. Replacing the values of these keys should present no issues, but removing them may cause Sass compilation issues. In these instances, you'll need to modify the Sass code that makes use of those values.

Functions

Bootstrap utilizes several Sass functions, but only a subset are applicable to general theming. We've included three functions for getting values from the color maps:

Copy

```
@function color($key: "blue") {
  @return map-get($colors, $key);
}

@function theme-color($key: "primary") {
  @return map-get($theme-colors, $key);
}

@function gray($key: "100") {
  @return map-get($grays, $key);
}
```

These allow you to pick one color from a Sass map much like how you'd use a color variable from v3.

Copy

```scss
.custom-element {
  color: gray("100");
  background-color: theme-color("dark");
}
```

We also have another function for getting a particular *level* of color from the `$theme-colors` map. Negative level values will lighten the color, while higher levels will darken.

Copy

```scss
@function theme-color-level($color-name: "primary", $level: 0) {
  $color: theme-color($color-name);
  $color-base: if($level > 0, #000, #fff);
  $level: abs($level);

  @return mix($color-base, $color, $level * $theme-color-interval);
}
```

In practice, you'd call the function and pass in two parameters: the name of the color from `$theme-colors` (e.g., primary or danger) and a numeric level.

Copy

```scss
.custom-element {
  color: theme-color-level(primary, -10);
}
```

Additional functions could be added in the future or your own custom Sass to create level functions for additional Sass maps, or even a generic one if you wanted to be more verbose.

Color contrast

One additional function we include in Bootstrap is the color contrast function, `color-yiq`. It utilizes the YIQ color space to automatically return a light (`#fff`) or dark (`#111`) contrast color based on the specified base color. This function is especially useful for mixins or loops where you're generating multiple classes.

For example, to generate color swatches from our `$theme-colors` map:

Copy

```scss
@each $color, $value in $theme-colors {
  .swatch-#{$color} {
    color: color-yiq($value);
  }
}
```

It can also be used for one-off contrast needs:

Copy

```scss
.custom-element {
  color: color-yiq(#000); // returns `color: #fff`
}
```

You can also specify a base color with our color map functions:

Copy

```scss
.custom-element {
  color: color-yiq(theme-color("dark")); // returns `color: #fff`
}
```

Sass options

Customize Bootstrap 4 with our built-in custom variables file and easily toggle global CSS preferences with new `$enable-*` Sass variables. Override a variable's value and recompile with `npm run test` as needed.

You can find and customize these variables for key global options in Bootstrap's `scss/_variables.scss` file.

Variable	Values	Description
`$spacer`	`1rem` (default), or any value > 0	Specifies the default spacer value to programmatically generate our spacer utilities.
`$enable-rounded`	`true` (default) or `false`	Enables predefined `border-radius` styles on various components.
`$enable-shadows`	`true` or `false` (default)	Enables predefined `box-shadow` styles on various components.
`$enable-gradients`	`true` or `false` (default)	Enables predefined gradients via `background-image` styles on various components.
`$enable-transitions`	`true` (default) or `false`	Enables predefined `transition`s on various components.
`$enable-prefers-reduced-motion-media-query`	`true` (default) or `false`	Enables the `prefers-reduced-motion` media query, which suppresses certain animations/transitions based on the users' browser/operating system preferences.
`$enable-hover-media-query`	`true` or `false` (default)	**Deprecated**
`$enable-grid-classes`	`true` (default) or `false`	Enables the generation of CSS classes for the grid system (e.g., `.container`, `.row`, `.col-md-1`, etc.).
`$enable-caret`	`true` (default) or `false`	Enables pseudo element caret on `.dropdown-toggle`.
`$enable-pointer-cursor-for-buttons`	`true` (default) or `false`	Add "hand" cursor to non-disabled button elements.
`$enable-print-styles`	`true` (default) or `false`	Enables styles for optimizing printing.
`$enable-responsive-font-sizes`	`true` or `false` (default)	Enables responsive font sizes.

Variable	Values	Description
`$enable-validation-icons`	`true` (default) or `false`	Enables `background-image` icons within textual inputs and some custom forms for validation states.
`$enable-deprecation-messages`	`true` or `false` (default)	Set to `true` to show warnings when using any of the deprecated mixins and functions that are planned to be removed in `v5`.

Color

Many of Bootstrap's various components and utilities are built through a series of colors defined in a Sass map. This map can be looped over in Sass to quickly generate a series of rulesets.

All colors

All colors available in Bootstrap 4, are available as Sass variables and a Sass map in `scss/_variables.scss` file. This will be expanded upon in subsequent minor releases to add additional shades, much like the grayscale palette we already include.

Blue Indigo Purple

Pink Red Orange

Yellow Green Teal

Cyan

Here's how you can use these in your Sass:

Copy

```
// With variable
.alpha { color: $purple; }

// From the Sass map with our `color()` function
.beta { color: color("purple"); }
```

Color utility classes are also available for setting `color` and `background-color`.

> In the future, we'll aim to provide Sass maps and variables for shades of each color as we've done with the grayscale colors below.

Theme colors

We use a subset of all colors to create a smaller color palette for generating color schemes, also available as Sass variables and a Sass map in Bootstraps's `scss/_variables.scss` file.

Primary Secondary Success

Danger Warning Info

Light Dark

Grays

An expansive set of gray variables and a Sass map in `scss/_variables.scss` for consistent shades of gray across your project. Note that these are “cool grays”, which tend towards a subtle blue tone, rather than neutral grays.

100

200

300

400

500

600

700

800

900

Within `scss/_variables.scss`, you’ll find Bootstrap’s color variables and Sass map. Here’s an example of the `$colors` Sass map:

Copy

```
$colors: (
  "blue": $blue,
  "indigo": $indigo,
  "purple": $purple,
  "pink": $pink,
  "red": $red,
  "orange": $orange,
  "yellow": $yellow,
  "green": $green,
  "teal": $teal,
  "cyan": $cyan,
  "white": $white,
  "gray": $gray-600,
  "gray-dark": $gray-800
) !default;
```

Add, remove, or modify values within the map to update how they’re used in many other components. Unfortunately at this time, not *every* component utilizes this Sass map. Future updates will strive to improve upon this. Until then, plan on making use of the `${color}` variables and this Sass map.

Components

Many of Bootstrap’s components and utilities are built with `@each` loops that iterate over a Sass map. This is especially helpful for generating variants of a component by our `$theme-colors` and creating responsive variants for each breakpoint. As you customize these Sass maps and recompile, you’ll automatically see your changes reflected in these loops.

Modifiers

Many of Bootstrap's components are built with a base-modifier class approach. This means the bulk of the styling is contained to a base class (e.g., `.btn`) while style variations are confined to modifier classes (e.g., `.btn-danger`). These modifier classes are built from the `$theme-colors` map to make customizing the number and name of our modifier classes.

Here are two examples of how we loop over the `$theme-colors` map to generate modifiers to the `.alert` component and all our `.bg-*` background utilities.

Copy

```
// Generate alert modifier classes
@each $color, $value in $theme-colors {
  .alert-#{$color} {
    @include alert-variant(theme-color-level($color, -10), theme-color-level($color,
-9), theme-color-level($color, 6));
  }
}

// Generate `.bg-*` color utilities
@each $color, $value in $theme-colors {
  @include bg-variant('.bg-#{$color}', $value);
}
```

Responsive

These Sass loops aren't limited to color maps, either. You can also generate responsive variations of your components or utilities. Take for example our responsive text alignment utilities where we mix an `@each` loop for the `$grid-breakpoints` Sass map with a media query include.

Copy

```
@each $breakpoint in map-keys($grid-breakpoints) {
  @include media-breakpoint-up($breakpoint) {
    $infix: breakpoint-infix($breakpoint, $grid-breakpoints);

    .text#{$infix}-left   { text-align: left !important; }
    .text#{$infix}-right  { text-align: right !important; }
    .text#{$infix}-center { text-align: center !important; }
  }
}
```

Should you need to modify your `$grid-breakpoints`, your changes will apply to all the loops iterating over that map.

CSS variables

Bootstrap 4 includes around two dozen CSS custom properties (variables) in its compiled CSS. These provide easy access to commonly used values like our theme colors, breakpoints, and primary font stacks when working in your browser's Inspector, a code sandbox, or general prototyping.

Available variables

Here are the variables we include (note that the `:root` is required). They're located in our `_root.scss` file.

Copy

Search...

Getting started
Introduction
Download
Contents
Browsers & devices
JavaScript
Theming
Build tools
Webpack
Accessibility

Layout
Content
Components
Utilities
Extend
Migration
About

```
:root {
  --blue: #007bff;
  --indigo: #6610f2;
  --purple: #6f42c1;
  --pink: #e83e8c;
  --red: #dc3545;
  --orange: #fd7e14;
  --yellow: #ffc107;
  --green: #28a745;
  --teal: #20c997;
  --cyan: #17a2b8;
  --white: #fff;
  --gray: #6c757d;
  --gray-dark: #343a40;
  --primary: #007bff;
  --secondary: #6c757d;
  --success: #28a745;
  --info: #17a2b8;
  --warning: #ffc107;
  --danger: #dc3545;
  --light: #f8f9fa;
  --dark: #343a40;
  --breakpoint-xs: 0;
  --breakpoint-sm: 576px;
  --breakpoint-md: 768px;
  --breakpoint-lg: 992px;
  --breakpoint-xl: 1200px;
  --font-family-sans-serif: -apple-system, BlinkMacSystemFont, "Segoe UI", Roboto,
"Helvetica Neue", Arial, sans-serif, "Apple Color Emoji", "Segoe UI Emoji", "Segoe UI
Symbol";
  --font-family-monospace: SFMono-Regular, Menlo, Monaco, Consolas, "Liberation Mono",
"Courier New", monospace;
}
```

Examples

CSS variables offer similar flexibility to Sass's variables, but without the need for compilation before being served to the browser. For example, here we're resetting our page's font and link styles with CSS variables.

Copy

```
body {
  font: 1rem/1.5 var(--font-family-sans-serif);
}
a {
  color: var(--blue);
}
```

Breakpoint variables

While we originally included breakpoints in our CSS variables (e.g., `--breakpoint-md`), **these are not supported in media queries**, but they can still be used *within* rulesets in media queries. These breakpoint variables remain in the compiled CSS for backward compatibility given they can be utilized by JavaScript. Learn more in the spec.

Here's an example of **what's not supported:**

Copy

```
@media (min-width: var(--breakpoint-sm)) {
  ...
}
```

And here's an example of **what is supported:**

Copy

```scss
@media (min-width: 768px) {
  .custom-element {
    color: var(--primary);
  }
}
```