Get familiar

We recommend getting familiar with Bootstrap first by reading through our [Getting Started Introduction page]({{< docsref "/getting-started/introduction" >}}). Once you've run through it, continue reading here for how to enable RTL.

You may also want to read up on the RTLCSS project, as it powers our approach to RTL.

{{< callout warning >}}

Experimental feature

The RTL feature is still **experimental** and will probably evolve according to user feedback. Spotted something or have an improvement to suggest? [Open an issue]({{< param repo >}}/issues/new), we'd love to get your insights. {{< /callout >}}

Required HTML

There are two strict requirements for enabling RTL in Bootstrap-powered pages.

1. Set `dir="rtl"` on the `<html>` element.
2. Add an appropriate `lang` attribute, like `lang="ar"`, on the `<html>` element.

From there, you'll need to include an RTL version of our CSS. For example, here's the stylesheet for our compiled and minified CSS with RTL enabled:

```
<link rel="stylesheet" href="{{< param "cdn.css_rtl" >}}" integrity="{{< param "cdn.css_rtl_hash" >}}" crossorigin="anonymous">
```

Starter template

You can see the above requirements reflected in this modified RTL starter template.

```
<!doctype html>
<html lang="ar" dir="rtl">
  <head>
    <!-- Required meta tags -->
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">

    <!-- Bootstrap CSS -->
    <link rel="stylesheet" href="{{< param "cdn.css_rtl" >}}" integrity="{{< param "cdn.css_rtl_hash" >}}" crossorigin="anonymous">

    <title>مرحبا بالعالم!</title>
  </head>
  <body>
    <h1>مرحبا بالعالم!</h1>

    <!-- Optional JavaScript; choose one of the two! -->

    <!-- Option 1: Bootstrap Bundle with Popper -->
```

```
    <script src="{{< param "cdn.js_bundle" >}}" integrity="{{< param
"cdn.js_bundle_hash" >}}" crossorigin="anonymous"></script>

    <!-- Option 2: Separate Popper and Bootstrap JS -->
    <!--
    <script src="{{< param "cdn.popper" >}}" integrity="{{< param "cdn.popper_hash"
>}}" crossorigin="anonymous"></script>
    <script src="{{< param "cdn.js" >}}" integrity="{{< param "cdn.js_hash" >}}"
crossorigin="anonymous"></script>
    -->
  </body>
</html>
```

RTL examples

Get started with one of our several [RTL examples]({{< docsref "/examples/#rtl" >}}).

Approach

Our approach to building RTL support into Bootstrap comes with two important decisions that impact how we write and use our CSS:

1. **First, we decided to build it with the RTLCSS project.** This gives us some powerful features for managing changes and overrides when moving from LTR to RTL. It also allows us to build two versions of Bootstrap from one codebase.
2. **Second, we've renamed a handful of directional classes to adopt a logical properties approach.** Most of you have already interacted with logical properties thanks to our flex utilities—they replace direction properties like `left` and `right` in favor `start` and `end`. That makes the class names and values appropriate for LTR and RTL without any overhead.

For example, instead of `.ml-3` for `margin-left`, use `.ms-3`.

Working with RTL, through our source Sass or compiled CSS, shouldn't be much different from our default LTR though.

Customize from source

When it comes to [customization]({{< docsref "/customize/sass" >}}), the preferred way is to take advantage of variables, maps, and mixins. This approach works the same for RTL, even if it's post-processed from the compiled files, thanks to how RTLCSS works.

Custom RTL values

Using RTLCSS value directives, you can make a variable output a different value for RTL. For example, to decrease the weight for `$font-weight-bold` throughout the codebase, you may use the `/*rtl: {value}*/` syntax:

```
$font-weight-bold: 700 #{/* rtl:600 */} !default;
```

Which would output to the following for our default CSS and RTL CSS:

```
/* bootstrap.css */
dt {
  font-weight: 700 /* rtl:600 */;
}

/* bootstrap.rtl.css */
dt {
  font-weight: 600;
}
```

Alternative font stack

In the case you're using a custom font, be aware that not all fonts support the non-Latin alphabet. To switch from Pan-European to Arabic family, you may need to use `/*rtl:insert: {value}*/` in your font stack to modify the names of font families.

For example, to switch from `Helvetica Neue Webfont` for LTR to `Helvetica Neue Arabic` for RTL, your Sass code look like this:

```
$font-family-sans-serif:
  Helvetica Neue #{"/* rtl:insert:Arabic */"},
  // Cross-platform generic font family (default user interface font)
  system-ui,
  // Safari for macOS and iOS (San Francisco)
  -apple-system,
  // Chrome < 56 for macOS (San Francisco)
  BlinkMacSystemFont,
  // Windows
  "Segoe UI",
  // Android
  Roboto,
  // Basic web fallback
  Arial,
  // Linux
  "Noto Sans",
  // Sans serif fallback
  sans-serif,
  // Emoji fonts
  "Apple Color Emoji", "Segoe UI Emoji", "Segoe UI Symbol", "Noto Color Emoji"
!default;
```

LTR and RTL at the same time

Need both LTR and RTL on the same page? Thanks to RTLCSS String Maps, this is pretty straightforward. Wrap your `@import`s with a class, and set a custom rename rule for RTLCSS:

```
/* rtl:begin:options: {
  "autoRename": true,
  "stringMap":[ {
    "name": "ltr-rtl",
```

```
    "priority": 100,
    "search": ["ltr"],
    "replace": ["rtl"],
    "options": {
      "scope": "*",
      "ignoreCase": false
    }
  } ]
} */
.ltr {
  @import "../node_modules/bootstrap/scss/bootstrap";
}
/*rtl:end:options*/
```

After running Sass then RTLCSS, each selector in your CSS files will be prepended by `.ltr`, and `.rtl` for RTL files. Now you're able to use both files on the same page, and simply use `.ltr` or `.rtl` on your components wrappers to use one or the other direction.

{{< callout warning >}}

Edge cases and known limitations

While this approach is understandable, please pay attention to the following:

1. When switching `.ltr` and `.rtl`, make sure you add `dir` and `lang` attributes accordingly.
2. Loading both files can be a real performance bottleneck: consider some [optimization]({{< docsref "/customize/optimize" >}}), and maybe try to load one of those files asynchronously.
3. Nesting styles this way will prevent our `form-validation-state()` mixin from working as intended, thus require you tweak it a bit by yourself. See #31223. {{< /callout >}}

The breadcrumb case

The [breadcrumb separator]({{< docsref "/components/breadcrumb" >}}/#changing-the-separator) is the only case requiring its own brand new variable— namely `$breadcrumb-divider-flipped` —defaulting to `$breadcrumb-divider`.

Additional resources

- RTLCSS
- RTL Styling 101