Layout

Options for structuring your pages with Bootstrap, including global styles, required scaffolding, grid system, and more.

Grid system

Bootstrap includes a powerful mobile-first grid system for building layouts of all shapes and sizes. It's based on a 12 column layout and has multiple tiers, one for each media query range. You can use it with Sass mixins or our predefined classes.

Contents

How it works

At a high level, here's how the grid system works:

- There are three major components—containers, rows, and columns.
- Containers— `.container` for fixed width or `.container-fluid` for full width—center your site's contents and help align your grid content.
- Rows are horizontal groups of columns that ensure your columns are lined up properly.
- Content should be placed within columns, and only columns may be immediate children of rows.
- Column classes indicate the number of columns you'd like to use out of the possible 12 per row. So if you want three equal-width columns, you'd use `.col-sm-4` .
- Column `width` s are set in percentages, so they're always fluid and sized relative to their parent element.
- Columns have horizontal `padding` to create the gutters between individual columns.
- There are five grid tiers, one for each responsive breakpoint: extra small, small, medium, large, and extra large.

- Grid tiers are based on minimum widths, meaning they apply to that one tier and all those above it (e.g., `.col-sm-4` applies to small, medium, large, and extra large devices).
- You can use predefined grid classes or Sass mixins for more semantic markup.

Sounds good? Great, let's move on to seeing all that in an example.

Quick start example

If you're using Bootstrap's compiled CSS, this the example you'll want to start with.

One of three columns
One of three columns
One of three columns

```
<div class="container">
  <div class="row">
    <div class="col-sm-4">
      One of three columns
    </div>
    <div class="col-sm-4">
      One of three columns
    </div>
    <div class="col-sm-4">
      One of three columns
    </div>
  </div>
</div>
```

The above example creates three equal-width columns on small, medium, large, and extra large devices using our predefined grid classes. Those columns are centered in the page with the parent `.container`.

Grid options

While Bootstrap uses `em`s or `rem`s for defining most sizes, `px`s are used for grid breakpoints and container widths. This is because the viewport width is in pixels and does not change with the font size.

See how aspects of the Bootstrap grid system work across multiple devices with a handy table.

	Extra small <576px	**Small** ≥576px	**Medium** ≥768px	**Large** ≥992px	**Extra large** ≥1200px
Grid behavior	Horizontal at all times	Collapsed to start, horizontal above breakpoints			
Max container width	None (auto)	540px	720px	960px	1140px
Class prefix	`.col-xs-`	`.col-sm-`	`.col-md-`	`.col-lg-`	`.col-xl-`
# of columns	12				
Gutter width	30px (15px on each side of a column)				
Nestable	Yes				
Offsets	Yes				
Column ordering	Yes				

Sass mixins

When using Bootstrap's source Sass files, you have the option of using Sass variables and mixins to create custom, semantic, and responsive page layouts. Our predefined grid classes use these same variables and mixins to provide a whole suite of ready-to-use classes for fast responsive layouts.

Variables

Variables and maps determine the number of columns, the gutter width, and the media query point at which to begin floating columns. We use these to generate the predefined grid classes documented above, as well as for the custom mixins listed below.

```
$grid-columns:      12;
$grid-gutter-width-base: 30px;

$grid-gutter-widths: (
  xs: $grid-gutter-width-base, // 30px
  sm: $grid-gutter-width-base, // 30px
  md: $grid-gutter-width-base, // 30px
  lg: $grid-gutter-width-base, // 30px
  xl: $grid-gutter-width-base  // 30px
)

$grid-breakpoints: (
  // Extra small screen / phone
  xs: 0,
  // Small screen / phone
  sm: 576px,
  // Medium screen / tablet
  md: 768px,
  // Large screen / desktop
  lg: 992px,
  // Extra large screen / wide desktop
  xl: 1200px
);

$container-max-widths: (
  sm: 540px,
  md: 720px,
  lg: 960px,
  xl: 1140px
);
```

Mixins

Mixins are used in conjunction with the grid variables to generate semantic CSS for individual grid columns.

```
// Creates a wrapper for a series of columns
@mixin make-row($gutters: $grid-gutter-widths) {
  @if $enable-flex {
    display: flex;
    flex-wrap: wrap;
  } @else {
    @include clearfix();
  }

  @each $breakpoint in map-keys($gutters) {
    @include media-breakpoint-up($breakpoint) {
      $gutter: map-get($gutters, $breakpoint);
      margin-right: ($gutter / -2);
      margin-left:  ($gutter / -2);
    }
  }
}

// Make the element grid-ready (applying everything but the width)
@mixin make-col-ready($gutters: $grid-gutter-widths) {
  position: relative;
  min-height: 1px; // Prevent collapsing

  // Prevent columns from becoming too narrow when at smaller grid tiers by
  // always setting `width: 100%;`. This works because we use `flex` values
  // later on to override this initial width.
  @if $enable-flex {
    width: 100%;
  }

  @each $breakpoint in map-keys($gutters) {
    @include media-breakpoint-up($breakpoint) {
      $gutter: map-get($gutters, $breakpoint);
      padding-right: ($gutter / 2);
      padding-left:  ($gutter / 2);
    }
  }
}

@mixin make-col($size, $columns: $grid-columns) {
  @if $enable-flex {
    flex: 0 0 percentage($size / $columns);
    // Add a `max-width` to ensure content within each column does not blow out
    // the width of the column. Applies to IE10+ and Firefox. Chrome and Safari
    // do not appear to require this.
    max-width: percentage($size / $columns);
  } @else {
    float: left;
    width: percentage($size / $columns);
```

```
  }
}

// Get fancy by offsetting, or changing the sort order
@mixin make-col-offset($size, $columns: $grid-columns) {
  margin-left: percentage($size / $columns);
}

@mixin make-col-push($size, $columns: $grid-columns) {
  left: if($size > 0, percentage($size / $columns), auto);
}

@mixin make-col-pull($size, $columns: $grid-columns) {
  right: if($size > 0, percentage($size / $columns), auto);
}
```

Example usage

You can modify the variables to your own custom values, or just use the mixins with their default values. Here's an example of using the default settings to create a two-column layout with a gap between.

See it in action in this rendered example.

```sass
.container {
  max-width: 60em;
  @include make-container();
}
.row {
  @include make-row();
}
.content-main {
  @include make-col-ready();

  @media (max-width: 32em) {
    @include make-col(6);
  }
  @media (min-width: 32.1em) {
    @include make-col(8);
  }
}
.content-secondary {
  @include make-col-ready();

  @media (max-width: 32em) {
    @include make-col(6);
  }
  @media (min-width: 32.1em) {
    @include make-col(4);
  }
}
```

```html
<div class="container">
  <div class="row">
    <div class="content-main">...</div>
    <div class="content-secondary">...</div>
  </div>
</div>
```

Predefined classes

In addition to our semantic mixins, Bootstrap includes an extensive set of prebuilt classes for quickly creating grid columns. It includes options for device-based column sizing, reordering columns, and more.

Example: Stacked-to-horizontal

Using a single set of `.col-md-*` grid classes, you can create a basic grid system that starts out stacked on mobile devices and tablet devices (the extra small to small range) before becoming horizontal on desktop (medium) devices. Place grid columns within any `.row`.

col-md-1

col-md-1

col-md-1

col-md-1

col-md-1

col-md-1

col-md-1

col-md-1

col-md-1

col-md-1

col-md-1

col-md-1

col-md-8

col-md-4

col-md-4

col-md-4

col-md-4

col-md-6

col-md-6

```
<div class="row">
  <div class="col-md-1">col-md-1</div>
  <div class="col-md-1">col-md-1</div>
  <div class="col-md-1">col-md-1</div>
  <div class="col-md-1">col-md-1</div>
  <div class="col-md-1">col-md-1</div>
  <div class="col-md-1">col-md-1</div>
  <div class="col-md-1">col-md-1</div>
  <div class="col-md-1">col-md-1</div>
  <div class="col-md-1">col-md-1</div>
  <div class="col-md-1">col-md-1</div>
  <div class="col-md-1">col-md-1</div>
  <div class="col-md-1">col-md-1</div>
</div>
<div class="row">
  <div class="col-md-8">col-md-8</div>
  <div class="col-md-4">col-md-4</div>
</div>
<div class="row">
  <div class="col-md-4">col-md-4</div>
  <div class="col-md-4">col-md-4</div>
  <div class="col-md-4">col-md-4</div>
</div>
<div class="row">
  <div class="col-md-6">col-md-6</div>
  <div class="col-md-6">col-md-6</div>
</div>
```

Example: Mobile and desktop

Don't want your columns to simply stack in smaller devices? Use the extra small and medium device grid classes by adding `.col-xs-*` and `.col-md-*` to your columns. See the example below for a better idea of how it all works.

.col-xs-12 .col-md-8

.col-xs-6 .col-md-4

.col-xs-6 .col-md-4 | .col-xs-6 .col-md-4

.col-xs-6 .col-md-4

.col-xs-6

.col-xs-6

```
<!-- Stack the columns on mobile by making one full-width and the other half-width -->
<div class="row">
  <div class="col-xs-12 col-md-8">.col-xs-12 .col-md-8</div>
  <div class="col-xs-6 col-md-4">.col-xs-6 .col-md-4</div>
</div>

<!-- Columns start at 50% wide on mobile and bump up to 33.3% wide on desktop -->
<div class="row">
  <div class="col-xs-6 col-md-4">.col-xs-6 .col-md-4</div>
  <div class="col-xs-6 col-md-4">.col-xs-6 .col-md-4</div>
  <div class="col-xs-6 col-md-4">.col-xs-6 .col-md-4</div>
</div>

<!-- Columns are always 50% wide, on mobile and desktop -->
<div class="row">
  <div class="col-xs-6">.col-xs-6</div>
  <div class="col-xs-6">.col-xs-6</div>
</div>
```

Example: Mobile, tablet, desktop

Build on the previous example by creating even more dynamic and powerful layouts with tablet `.col-sm-*` classes.

.col-xs-12 .col-sm-6 .col-md-8

.col-xs-6 .col-md-4

.col-xs-6 .col-sm-4

.col-xs-6 .col-sm-4

.col-xs-6 .col-sm-4

```
<div class="row">
  <div class="col-xs-12 col-sm-6 col-md-8">.col-xs-12 .col-sm-6 .col-md-8</div>
  <div class="col-xs-6 col-md-4">.col-xs-6 .col-md-4</div>
</div>
<div class="row">
  <div class="col-xs-6 col-sm-4">.col-xs-6 .col-sm-4</div>
  <div class="col-xs-6 col-sm-4">.col-xs-6 .col-sm-4</div>
  <!-- Optional: clear the XS cols if their content doesn't match in height -->
  <div class="clearfix hidden-sm-up"></div>
  <div class="col-xs-6 col-sm-4">.col-xs-6 .col-sm-4</div>
</div>
```

Example: Column wrapping

If more than 12 columns are placed within a single row, each group of extra columns will, as one unit, wrap onto a new line.

.col-xs-9

.col-xs-4
Since 9 + 4 = 13 > 12, this 4-column-wide div gets wrapped onto a new line as one contiguous unit.

.col-xs-6
Subsequent columns continue along the new line.

```
<div class="row">
  <div class="col-xs-9">.col-xs-9</div>
  <div class="col-xs-4">.col-xs-4<br>Since 9 + 4 = 13 &gt; 12, this 4-column-wide div
gets wrapped onto a new line as one contiguous unit.</div>
  <div class="col-xs-6">.col-xs-6<br>Subsequent columns continue along the new line.
</div>
</div>
```

Example: Responsive column resets

With the four tiers of grids available you're bound to run into issues where, at certain breakpoints, your columns don't clear quite right as one is taller than the other. To fix that, use a combination of a `.clearfix` and our responsive utility classes.

.col-xs-6 .col-sm-3	.col-xs-6 .col-sm-3

.col-xs-6 .col-sm-3	.col-xs-6 .col-sm-3

```
<div class="row">
  <div class="col-xs-6 col-sm-3">.col-xs-6 .col-sm-3</div>
  <div class="col-xs-6 col-sm-3">.col-xs-6 .col-sm-3</div>

  <!-- Add the extra clearfix for only the required viewport -->
  <div class="clearfix hidden-sm-up"></div>

  <div class="col-xs-6 col-sm-3">.col-xs-6 .col-sm-3</div>
  <div class="col-xs-6 col-sm-3">.col-xs-6 .col-sm-3</div>
</div>
```

In addition to column clearing at responsive breakpoints, you may need to **reset offsets, pushes, or pulls**. See this in action in the grid example.

.col-sm-5 .col-md-6
.col-sm-5 .offset-sm-2 .col-md-6 .offset-md-0

.col.col-sm-6.col-md-5.col-lg-6
.col-sm-6 .col-md-5 .offset-md-2 .col-lg-6 .offset-lg-0

```
<div class="row">
  <div class="col-sm-5 col-md-6">.col-sm-5 .col-md-6</div>
  <div class="col-sm-5 offset-sm-2 col-md-6 offset-md-0">.col-sm-5 .offset-sm-2 .col-
md-6 .offset-md-0</div>
</div>

<div class="row">
  <div class="col-sm-6 col-md-5 col-lg-6">.col.col-sm-6.col-md-5.col-lg-6</div>
  <div class="col-sm-6 col-md-5 offset-md-2 col-lg-6 offset-lg-0">.col-sm-6 .col-md-5
.offset-md-2 .col-lg-6 .offset-lg-0</div>
</div>
```

Example: Offsetting columns

Move columns to the right using `.offset-md-*` classes. These classes increase the left margin of a column by `*` columns. For example, `.offset-md-4` moves `.col-md-4` over four columns.

.col-md-4

.col-md-4 .offset-md-4

.col-md-3 .offset-md-3

.col-md-3 .offset-md-3

.col-md-6 .offset-md-3

```
<div class="row">
  <div class="col-md-4">.col-md-4</div>
  <div class="col-md-4 offset-md-4">.col-md-4 .offset-md-4</div>
</div>
<div class="row">
  <div class="col-md-3 offset-md-3">.col-md-3 .offset-md-3</div>
  <div class="col-md-3 offset-md-3">.col-md-3 .offset-md-3</div>
</div>
<div class="row">
  <div class="col-md-6 offset-md-3">.col-md-6 .offset-md-3</div>
</div>
```

Example: Nesting columns

To nest your content with the default grid, add a new `.row` and set of `.col-sm-*` columns within an existing `.col-sm-*` column. Nested rows should include a set of columns that add up to 12 or fewer (it is not required that you use all 12 available columns).

Level 1: .col-sm-9

Level 2: .col-xs-8 .col-sm-6

Level 2: .col-xs-4 .col-sm-6

```
<div class="row">
  <div class="col-sm-9">
    Level 1: .col-sm-9
    <div class="row">
      <div class="col-xs-8 col-sm-6">
        Level 2: .col-xs-8 .col-sm-6
      </div>
      <div class="col-xs-4 col-sm-6">
        Level 2: .col-xs-4 .col-sm-6
      </div>
    </div>
  </div>
</div>
```

Example: Column ordering

Easily change the order of our built-in grid columns with `.push-md-*` and `.pull-md-*` modifier classes.

.col-md-9 .push-md-3
.col-md-3 .pull-md-9

```
<div class="row">
  <div class="col-md-9 push-md-3">.col-md-9 .push-md-3</div>
  <div class="col-md-3 pull-md-9">.col-md-3 .pull-md-9</div>
</div>
```

Customizing the grid

Using our built-in grid Sass variables and maps, it's possible to completely customize the predefined grid classes. Change the number of tiers, the media query dimensions, and the container widths—then recompile.

Columns and gutters

The number of grid columns and their horizontal padding (aka, gutters) can be modified via Sass variables. `$grid-columns` is used to generate the widths (in percent) of each individual column while `$grid-gutter-widths` allows breakpoint-specific widths that are divided evenly across `padding-left` and `padding-right` for the column gutters.

```
$grid-columns:               12 !default;
$grid-gutter-width-base:     30px !default;
$grid-gutter-widths: (
  xs: $grid-gutter-width-base,
  sm: $grid-gutter-width-base,
  md: $grid-gutter-width-base,
  lg: $grid-gutter-width-base,
  xl: $grid-gutter-width-base
) !default;
```

Grid tiers

Moving beyond the columns themselves, you may also customize the number of grid tiers. If you wanted just three grid tiers, you'd update the `$grid-breakpoints` and `$container-max-widths` to something like this:

```
$grid-breakpoints: (
  sm: 480px,
  md: 768px,
  lg: 1024px
);

$container-max-widths: (
  sm: 420px,
  md: 720px,
  lg: 960px
);
```

When making any changes to the Sass variables or maps, you'll need to save your changes and recompile. Doing so will out a brand new set of predefined grid classes for column widths, offsets, pushes, and pulls. Responsive visibility utilities will also be updated to use the custom breakpoints.