CSS Grid - Complete Guide

What is CSS Grid?

CSS Grid is a **two-dimensional layout module** that allows you to create complex layouts with rows and columns simultaneously. Unlike Flexbox (which is one-dimensional), Grid gives you complete control over both horizontal and vertical alignment.

Grid Terminology

1. Grid Line

- **Definition**: Grid lines are horizontal and vertical lines that run through the entire CSS grid
- **Purpose**: These lines separate elements from one another
- **Analogy**: Like the lines that separate columns and rows in a table

```
.container {
  display: grid;
  grid-template-columns: 100px 100px 100px; /* Creates 4 vertical grid lines
*/
  grid-template-rows: 50px 50px; /* Creates 3 horizontal grid lines */
}
```

Visual Representation:

```
Grid Lines Example:
    |     |     |     |  ← Vertical grid lines (1-4)
----+-----+-----+-----+  ← Horizontal grid line 1
    |  A  |  B  |  C  |
----+-----+-----+-----+  ← Horizontal grid line 2
    |  D  |  E  |  F  |
----+-----+-----+-----+  ← Horizontal grid line 3
```

2. Grid Tracks

- **Definition**: The space between any two consecutive grid lines
- **Types**: Can be either a row track or a column track
- **Analogy**: The actual space/area between the lines

```css
.container {
  display: grid;
  grid-template-columns: 1fr 2fr 1fr; /* Creates 3 column tracks */
  grid-template-rows: 100px auto; /* Creates 2 row tracks */
}
```

3. Grid Cells

- **Definition**: The space present between any four intersecting grid lines
- **Importance**: It's the smallest unit in CSS Grid
- **Analogy**: Like individual cells in a table

```css
.container {
  display: grid;
  grid-template-columns: repeat(3, 1fr);
  grid-template-rows: repeat(2, 100px);
  /* This creates 6 grid cells (3×2) */
}
```

4. Grid Areas

- **Definition**: A collection of grid cells that form a rectangular area
- **Flexibility**: Can be a single cell or multiple cells
- **Usage**: Perfect for creating named layout sections

```css
.container {
  display: grid;
  grid-template-areas:
    "header header header"
    "sidebar main main"
    "footer footer footer";
}

.header { grid-area: header; }
.sidebar { grid-area: sidebar; }
.main { grid-area: main; }
.footer { grid-area: footer; }
```

5. Grid Columns

- **Definition**: The space between any two adjacent vertical grid lines
- **Similarity**: Similar to grid tracks but specifically vertical
- **Analogy**: Columns in a table

6. Grid Rows

- **Definition**: The space between any two adjacent horizontal grid lines
- **Similarity**: Similar to grid tracks but specifically horizontal
- **Analogy**: Rows in a table

7. Gutters

- **Definition**: Space between adjacent rows or columns
- **Purpose**: Creates visual separation between grid items

- **Implementation**: Using the `gap` property

```css
.container {
  display: grid;
  grid-template-columns: repeat(3, 1fr);
  gap: 20px; /* 20px gutter between all items */
  /* Or separately: */
  row-gap: 10px;
  column-gap: 20px;
}
```

Grid Container Properties

1. `display`

Purpose: Defines the element as a grid container

```css
.container {
  display: grid; /* Block-level grid */
}

.inline-container {
  display: inline-grid; /* Inline-level grid */
}
```

Example:

```html
<div class="container">
  <div class="item">1</div>
  <div class="item">2</div>
  <div class="item">3</div>
</div>
```

2. `grid-template-columns`

Purpose: Creates columns inside the grid

```css
/* Fixed sizes */
.container {
  grid-template-columns: 100px 200px 100px;
}

/* Flexible sizes */
.container {
  grid-template-columns: 1fr 2fr 1fr; /* Fractional units */
}
/* Mixed sizes */
.container {
  grid-template-columns: 200px 1fr auto;
}
/* Repeat function */
.container {
  grid-template-columns: repeat(3, 1fr); /* Same as: 1fr 1fr 1fr */
}
/* MinMax function */
.container {
  grid-template-columns: repeat(3, minmax(100px, 1fr));
}
```

3. `grid-template-rows`

Purpose: Creates rows inside the grid

```css
/* Fixed heights */
.container {
  grid-template-rows: 100px 200px 100px;
}
/* Auto-sizing */
.container {
  grid-template-rows: auto 1fr auto;
}
/* Repeat with different sizes */
.container {
  grid-template-rows: repeat(2, minmax(50px, auto));
}
```

- 1fr means if i have width of 1000px i want divide in 3 equal parts then 1fr = 1000/3, they divide internally and makes partition.

4. `grid-template`

Purpose: Shorthand for `grid-template-rows` and `grid-template-columns`

```
/* Syntax: rows / columns */
.container {
  grid-template: 100px 200px / 1fr 2fr 1fr;
}

/* With named areas */
.container {
  grid-template:
    "header header" 100px
    "sidebar main" 1fr
    "footer footer" 50px
    / 200px 1fr;
}
```

5. `gap` (formerly `grid-gap`)

Purpose: Creates gutters between grid items

```
/* Same gap for rows and columns */
.container {
  gap: 20px;
}

/* Different gaps */
.container {
  gap: 10px 20px; /* row-gap column-gap */
}

/* Individual properties */
.container {
  row-gap: 10px;
  column-gap: 20px;
}
```

6. `justify-items`

Purpose: Aligns items horizontally within their grid cells

```css
.container {
  justify-items: start;    /* Left align */
  justify-items: end;      /* Right align */
  justify-items: center;   /* Center align */
  justify-items: stretch;  /* Fill width (default) */
}
```

7. `align-items`

Purpose: Aligns items vertically within their grid cells

```css
.container {
  align-items: start;    /* Top align */
  align-items: end;      /* Bottom align */
  align-items: center;   /* Center align */
  align-items: stretch;  /* Fill height (default) */
}
```

8. `place-items`

Purpose: Shorthand for `align-items` and `justify-items`

```css
.container {
  place-items: center;           /* Both center */
  place-items: start end;        /* align-items justify-items */
  place-items: center stretch;   /* center vertically, stretch horizontally
*/
}
```

9. `justify-content`

Purpose: Aligns the entire grid horizontally within the container

```css
.container {
  justify-content: start;         /* Left align grid */
  justify-content: end;           /* Right align grid */
  justify-content: center;        /* Center grid */
  justify-content: stretch;       /* Stretch to fill */
  justify-content: space-around;  /* Equal space around */
  justify-content: space-between; /* Space between items */
  justify-content: space-evenly;  /* Equal space everywhere */
}
```

10. `align-content`

Purpose: Aligns the entire grid vertically within the container

```css
.container {
  align-content: start;         /* Top align grid */
  align-content: end;           /* Bottom align grid */
  align-content: center;        /* Center grid */
  align-content: stretch;       /* Stretch to fill */
  align-content: space-around;  /* Equal space around */
  align-content: space-between; /* Space between items */
  align-content: space-evenly;  /* Equal space everywhere */
}
```

11. `place-content`

Purpose: Shorthand for `align-content` and `justify-content`

```css
.container {
  place-content: center;              /* Both center */
  place-content: start end;           /* align-content justify-content */
  place-content: space-between center; /* space-between vertically, center
horizontally */
}
```

12. `grid-auto-flow`

Purpose: Defines how the auto-placement algorithm works

```css
.container {
  grid-auto-flow: row;          /* Fill rows first (default) */
  grid-auto-flow: column;       /* Fill columns first */
  grid-auto-flow: row dense;    /* Fill holes in rows */
  grid-auto-flow: column dense; /* Fill holes in columns */
}
```

Example:

```css
.container {
  display: grid;
  grid-template-columns: repeat(3, 1fr);
  grid-auto-flow: column; /* Items flow in columns instead of rows */
}
```

13. `grid-auto-rows`

Purpose: Defines the size of auto-generated rows

```css
.container {
  grid-auto-rows: 100px;                /* Fixed height */
  grid-auto-rows: minmax(50px, auto); /* Minimum 50px, grows as needed */
  grid-auto-rows: 1fr;                  /* Take available space */
}
```

14. `grid-auto-columns`

Purpose: Defines the size of auto-generated columns

```css
.container {
  grid-auto-columns: 200px;              /* Fixed width */
  grid-auto-columns: minmax(100px, 1fr); /* Minimum 100px, grows as needed */
  grid-auto-columns: max-content;        /* Size to content */
}
```

Grid Item Properties

1. `grid-column-start` / `grid-column-end`

```css
.item {
  grid-column-start: 1;
  grid-column-end: 3; /* Spans from line 1 to line 3 */
}
```

2. `grid-column` (shorthand)

```css
.item {
  grid-column: 1 / 3;      /* From line 1 to line 3 */
  grid-column: 1 / span 2; /* Start at line 1, span 2 columns */
  grid-column: span 2;     /* Span 2 columns from auto position */
}
```

3. `grid-row-start` / `grid-row-end`

```css
.item {
  grid-row-start: 2;
  grid-row-end: 4; /* Spans from row line 2 to line 4 */
}
```

4. `grid-row` (shorthand)

```css
.item {
  grid-row: 2 / 4;      /* From line 2 to line 4 */
  grid-row: 1 / span 3; /* Start at line 1, span 3 rows */
}
```

5. `grid-area`

```css
/* Using line numbers */
.item {
  grid-area: 1 / 1 / 3 / 3; /* row-start / col-start / row-end / col-end */
}

/* Using named areas */
.item {
  grid-area: header; /* Use named grid area */
}
```

Practical Examples

Example 1: Basic Grid Layout

```html
<div class="grid-container">
  <div class="item">1</div>
  <div class="item">2</div>
  <div class="item">3</div>
  <div class="item">4</div>
  <div class="item">5</div>
  <div class="item">6</div>
</div>
```

```css
.grid-container {
  display: grid;
  grid-template-columns: repeat(3, 1fr);
  grid-template-rows: repeat(2, 150px);
  gap: 20px;
  padding: 20px;
}

.item {
  background-color: #3498db;
  color: white;
  display: flex;
  align-items: center;
  justify-content: center;
  font-size: 24px;
}
```

Example 2: Responsive Grid with Named Areas

```html
<div class="layout">
  <header class="header">Header</header>
  <nav class="sidebar">Sidebar</nav>
  <main class="main">Main Content</main>
  <footer class="footer">Footer</footer>
</div>
```

```css
.layout {
  display: grid;
  min-height: 100vh;
  grid-template-areas:
    "header header"
    "sidebar main"
    "footer footer";
  grid-template-columns: 200px 1fr;
  grid-template-rows: auto 1fr auto;
  gap: 10px;
}

.header {
  grid-area: header;
  background: #2c3e50;
  color: white;
  padding: 1rem;
}

.sidebar {
  grid-area: sidebar;
  background: #34495e;
  color: white;
  padding: 1rem;
}

.main {
  grid-area: main;
  background: #ecf0f1;
  padding: 1rem;
}

.footer {
  grid-area: footer;
  background: #95a5a6;
  padding: 1rem;
}

/* Responsive */
@media (max-width: 768px) {
  .layout {
    grid-template-areas:
      "header"
      "main"
      "sidebar"
      "footer";
    grid-template-columns: 1fr;
  }
}
```

Example 3: Complex Grid with Spanning Items

```css
.gallery {
  display: grid;
  grid-template-columns: repeat(4, 1fr);
  grid-auto-rows: 200px;
  gap: 15px;
}

.item-1 { grid-column: span 2; grid-row: span 2; }
.item-2 { grid-column: span 2; }
.item-3 { grid-row: span 2; }
.item-4 { grid-column: span 3; }
```

Advanced Grid Functions

1. `repeat()`

```css
/* Basic repeat */
grid-template-columns: repeat(3, 1fr);

/* Repeat with different sizes */
grid-template-columns: repeat(2, 100px 200px); /* 100px 200px 100px 200px */

/* Auto-repeat */
grid-template-columns: repeat(auto-fit, minmax(250px, 1fr));
grid-template-columns: repeat(auto-fill, minmax(200px, 1fr));
```

2. `minmax()`

```css
/* Minimum 100px, maximum 1fr */
grid-template-columns: minmax(100px, 1fr) 200px minmax(50px, 300px);

/* Responsive columns */
grid-template-columns: repeat(auto-fit, minmax(250px, 1fr));
```

3. `fit-content()`

```
grid-template-columns: fit-content(300px) 1fr fit-content(200px);
```

Grid vs Flexbox - When to Use What?

Use CSS Grid when:

- You need two-dimensional layouts (rows AND columns)
- You want to create complex layouts
- You need precise control over item placement
- You're building overall page layouts

Use Flexbox when:

- You need one-dimensional layouts (either row OR column)
- You want to distribute space among items in a single direction
- You're aligning items within a container
- You're building component layouts

Browser Support

CSS Grid is supported in all modern browsers. For older browser support, use feature queries:

```css
.container {
  /* Fallback for older browsers */
  display: flex;
  flex-wrap: wrap;
}

@supports (display: grid) {
  .container {
    display: grid;
    grid-template-columns: repeat(3, 1fr);
  }
}
```

Common Grid Patterns

1. Equal Height Cards

```css
.card-grid {
  display: grid;
  grid-template-columns: repeat(auto-fit, minmax(300px, 1fr));
  gap: 2rem;
}
```

2. Sidebar Layout

```css
.sidebar-layout {
  display: grid;
  grid-template-columns: 250px 1fr;
  min-height: 100vh;
}
```

3. Holy Grail Layout

```css
.holy-grail {
  display: grid;
  grid-template:
    "header header header" auto
    "nav main aside" 1fr
    "footer footer footer" auto
    / 200px 1fr 200px;
  min-height: 100vh;
}
```