CONTENTS

RELATED

// Tutorial //

Getting Started with Less

Updated on September 15, 2020

CSS

By Nicholas Cerminara
Developer and author at DigitalOcean.

Introduction

If you're a front-end developer **who has yet to take the leap** into using a preprocessor, this is the article for you. I'll take you from beginning to end on getting

POPULAR TOPICS

Ubuntu
Linux Basics
JavaScript
React
Python
Security
Apache

up and running with Less in no time.

Below you'll find examples, starters, and, most importantly, a super easy and professional way to compile your Less into CSS. Less has a ton of features, so I'll try and cover only the most important pieces to point you in the right direction. Some of things we'll cover:

- A brief overview
- Compiling Less in multiple ways
- A simple, yet featured pack Gulp Starter Kit for you to test and repurpose
- A explanation of the easiest way to use Gulp to compile your Less to CSS on save, lint it for errors, combine it into one file, auto-prefix it, and then minify it.
- A ton of Less features and syntax

Less is More

Less is a CSS preprocessor with a CSS-like syntax. It looks identical to CSS, but has a ton of extra cool features that can speed development up like crazy. Below is a small example of what some Less code looks like to give you a general idea.

```
@import 'bootstrap/bootstrap';

@color: rgb(255, 69, 69);

a {
    .fancy-text();
    color: @color;
    transition: all 225ms ease 0ms;

    &:hover,
    &:focus {
        color: lighten(@link-color, 50%);
    }
}

.fancy-text() {
```


MySQL
Databases
Docker
Kubernetes
Ebooks
Browse all topic tags

ALL TUTORIALS →

QUESTIONS ▼
Q&A
Ask a question
DigitalOcean Product Docs
DigitalOcean Support

EVENTS ▼
Tech Talks
Hacktoberfest
Deploy

GET INVOLVED ▼
Community Newsletter
Hollie's Hub for Good
Write for DOnations
Community tools and integrations
Hatch Startup program

CREATE YOUR FREE COMMUNITY ACCOUNT! →

```
        font-family: 'Some fancy font';
        font-style: italic;
        font-weight: 700;
    }
```

The above code is *short* and **sweet**, but, most importantly, it is packed full of a ton awesome capabilities.

About the Syntax

The code sample above does a few things:

- It imports Bootstrap CSS files with all their awesome Less mixins
- On hover and focus of the `a` tag, it will brighten by 50%
- The `a` tag inherits all of the properties of `.fancy-text()`

You can see from just this example how efficient writing in Less can be for front-end developers.

What can happen on save

Since Less is a preprocessor, it has to be compiled into CSS before it can be used. During this process, you can inject your own tasks to happen at the same time. Some of things are (and that will be showed later):

- Checks for errors
- Autoprefixes for older browsers
- Combines (concats) all files into one
- Minify the code
- Livereload a webpage (not part of this demo)

There's a million different ways to do this and tailor it for your app site's unique needs.

How to Compile Less into CSS

In my personal opinion, picking up the syntax of Less is the easy part - especially since it feels like a natural extension of CSS. I believe the **biggest hurdle** for developers with Less is the compiling component. Between people using different local / server environments, being scared of the command line, or just the hurdle caused by learning a task-runner, integrating Less or any preprocessor can be quite troublesome without some guidance.

Using Gulp

Gulp is a Javascript based task runner. In layman terms, it helps you automate your development workflow. Things like minifying CSS, moving files to production folders, and even compiling Less to CSS. This is my preferred way of compiling Less.

Need a Gulp intro? Scotch.io guest writer, Justin Rexroad, wrote an amazing blog post on Getting Started with Gulp. He teaches you some really important things and how to customize it.

I'm going try and make this dead simple by removing all of the overhead of Less and Gulp for you. This way you can dive right in. **I've already configured a dead simple Gulp setup with Less configured for you**. You can check it out here.

First install Node.js by downloading it here. Then, **here's what you need to do to get started** (this is for Mac users, Windows will be nearly identical):

- Install Gulp globally (only need to do once, ever)
- Clone the our Gulp and Less Starter Kit
- Install configured dependencies
- Start Gulp

```
$ npm install --global gulp
$ git clone git@github.com:scotch-io/gulp-and-less-starter-kit.git
$ cd gulp-and-less-starter-kit
$ npm install
$ gulp
```

Copy

That's it! If you want to tweak the setup of the example, stop gulp with `ctrl c`, tweak the gulp.json file, and just rerun the `gulp` command.

Quick Tips About the Starter

For Less, the starter has these awesome features:

- Watches for Less changes on save
- Checks for Less errors and outputs them without you having to rerun Gulp
- Autoprefixes for all legacy browsers
- Combines all CSS into one big and sexy minified file
- Includes Bootstrap (Less version) and their mixins

Since this is a task runner, it also does these common commands for your Javascript files:

- Automatically compiles all jQuery libraries into one big JS file. Just add to the `lib` folder
- Lints custom scripts for errors. Just add to the `js` folder
- Combines all custom scripts into one js file

Video Demo

If you're struggling, check out the demo video below. This demo assumes you've already installed Node.js via the download link.

Other Methods to Compile Less

There's a bunch of ways to compile Less into CSS. The Gulp starter above is great because it does it in a way that allows you to do a lot more in the future. It's also easy to share the Gulp file with other developers so they don't have to figure out compiling on their own. Plus, it's free!

If you want to check out other ways though, look into these resources:

- Command-line with Rhino

- Client Side Compile
- Via Sublime
- With Grunt
- SimpLess
- WinLess
- CodeKit

Now that we've passed the hurdle of compiling Less and integrating it into out workflow, let's finally start coding!

Nesting

Right off the bat, nesting is one of the Less's coolest features. Nesting lets you do more and write less. Nesting allows you to cascade your css properties with inheriting children.

Here's an example of basic nesting:

```
/* Less Code */
h1 {
    color: rgb(255, 123, 123);
    font-weight: 300;

    span {
        font-weight: 600;
        color: rgb(255, 69, 69) ;
    }

    &:hover {
        color: rgb(200, 0, 0);
    }

    &:hover span {
        color: rgb(50, 50, 58);
    }
}
```

Copy

```
/* Compiled CSS code */
h1 {
    color: #ff7b7b;
    font-weight: 300;
}
h1 span {
    font-weight: 600;
    color: #ff4545;
}
h1:hover {
    color: #c80000;
}
h1:hover span {
    color: #32323a;
}
```

Copy

See the Pen Less Nesting Example by Nicholas Cerminara (@ncerminara) on CodePen.

Variables

Without a doubt, CSS variables will be a thing of the future. There's already a spec for them. With Less, you can start using this concept now.

```
/* Less Code */
@heading-color: rgb(99, 200, 200);
h1 {
    color: @heading-color;
    font-weight: 300;
}
```

Copy

```
/* Compiled CSS code */
h1 {
    color: #63c8c8;
```

Copy

```
        font-weight: 300;
    }
```

See the Pen Less Variables Example by Nicholas Cerminara (@ncerminara) on CodePen.

Importing

Importing works very similar to regular CSS importing. The main difference is you way you treat it though. With raw CSS, you typically don't want to do too many imports for performance reasons. With Less, it's actually really smart to treat them like includes. Some of the benefits:

- Easier to collaborate with others
- Variables are shared across imports
- Creates (almost forces) a modular and dry approach to front-end development

For example, with Twitter's Bootstrap and Less, you only have to include one Less file, yet you get a ton of well organized goodies.

Check out that what that file looks like below:

Copy

```
// Core variables and mixins
@import "variables.less";
@import "mixins.less";

// Reset and dependencies
@import "normalize.less";
@import "print.less";
@import "glyphicons.less";

// Core CSS
@import "scaffolding.less";
@import "type.less";
@import "code.less";
@import "grid.less";
```

```
    @import "tables.less";
    @import "forms.less";
    @import "buttons.less";

    // Components
    @import "component-animations.less";
    @import "dropdowns.less";
    @import "button-groups.less";
    @import "input-groups.less";
    @import "navs.less";
    @import "navbar.less";
    @import "breadcrumbs.less";
    @import "pagination.less";
    @import "pager.less";
    @import "labels.less";
    @import "badges.less";
    @import "jumbotron.less";
    @import "thumbnails.less";
    @import "alerts.less";
    @import "progress-bars.less";
    @import "media.less";
    @import "list-group.less";
    @import "panels.less";
    @import "responsive-embed.less";
    @import "wells.less";
    @import "close.less";

    // Components w/ JavaScript
    @import "modals.less";
    @import "tooltip.less";
    @import "popovers.less";
    @import "carousel.less";

    // Utility classes
    @import "utilities.less";
    @import "responsive-utilities.less";
```

Mixins

Mixins are exactly what they sound like. You can literally mix and match CSS classes with each other. For example, with Bootstrap you might have put all these classes in your HTML to build a **big sexy button**:

```
<a href="https://scotch.io" class="btn btn-success btn-lg btn-b
```
Copy

That's one approach, but with Less, you could do something like this:

```
.big-sexy-button {
    .btn;
    .btn-success;
    .btn-lg;
    .btn-block;
}
```
Copy

Here, you are remaking that **big sexy button** by only adding the class `big-sexy-button`. I love this concept. You are literally building and combining classes from existing ones. Your HTML stays super clean, and your CSS remains simple.

Mixins as a Function

Mixins can also be functions. This way you can pass a variable or parameter to your added mixin. For example, check this mixin out that puts a border only on the top of a div.

```
/* Less Code */
div {
    .border-top-radius(125px);

    background: rgb(255, 255, 255);
    display: inline-block;
    padding: 50px;
}
.border-top-radius(@radius) {
    border-top-right-radius: @radius;
```
Copy

```
    border-top-left-radius: @radius;
}
```

```
/* Compiled CSS code */
div {
    border-top-right-radius: 125px;
    border-top-left-radius: 125px;
    background: #ffffff;
    display: inline-block;
    padding: 50px;
}
```

Copy

See the Pen Less Mixins Example by Nicholas Cerminara (@ncerminara) on CodePen.

You probably see where this is going. You can probably want to write your own custom mixin library or use an existing one to help speed things up. These are the two big ones:

- Less Hat
- Less Elements

Operations

Operations bring math and calculations to your CSS. You can do operations on the following data types:

- A Number
- A Color
- A Variable

Check out the super basic code snippet below to see it in action:

```
/* Number */
h1 {
```

Copy

```
    margin-bottom: 10px - 5px;
    margin-bottom: 10px - 5;
}

/* Color */
h1 {
    color: #888888 / 2;
}

/* Variable */
@h1-default-margin-bottom:  25px;
h1.tight {
    margin-bottom: @h1-default-margin-bottom - 10px
}
```

Scope

In layman terms, scope is a fancy way of saying that you can override variables on a per class basis without messing up all your global variable settings.

A good use case for this is overriding a default style in a special case. You can also put your new class variables anywhere in the class (first or last) - scope will behave the same.

Check out these snippets that explain it:

```
@default-color: black;

/* Makes a tags red */
a {
    color: @default-color;
    @default-color: red;
}

/* h1s are still black */
h1 {
```

Copy

```
        color: @default-color;
    }
```

See the Pen Less Scope Example by Nicholas Cerminara (@ncerminara) on CodePen.

Conclusion

I've been familiar with all of the benefits of CSS preprocessors for quite a while. Admittedly, at first I was reluctant to push it into my production environment in fears of making it difficult for people who were collaborating, creating unnecessary overhead for myself, or simply burning too much time on configuration. I'm a strong believer of using the right tool for the job and never over-engineering anything. Less is actually none of these.

I can't stress enough the benefits gained while using it. I really hope this tutorial combined with the provided Gulp Starter for Less is enough to get you going. Taking that leap in using preprocessors on a regular basis is totally worth it in every way.

Finally, if you're interested in also learning SASS, Ken Wheeler wrote an excellent Post on it here.

Sign up →

About the authors

Nicholas Cerminara Author

Developer and author at DigitalOcean.

Still looking for an answer?

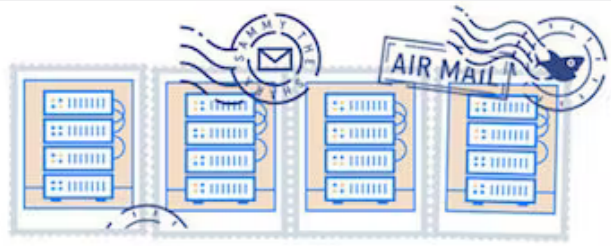
Ask a question | Search for more help

Comments

Leave a comment

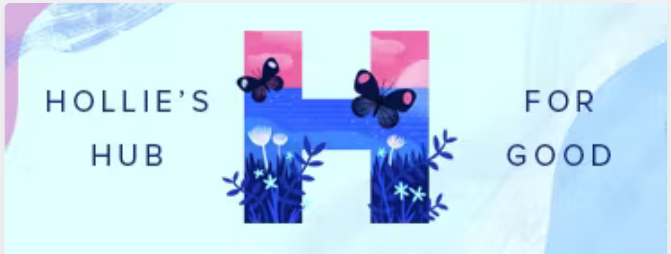
Leave a comment...

Login to Comment



GET OUR BIWEEKLY NEWSLETTER

Sign up for Infrastructure as a Newsletter.



HOLLIE'S HUB FOR GOOD

Working on improving health and education, reducing inequality, and spurring economic growth? We'd like to help.

BECOME A CONTRIBUTOR

You get paid; we donate to tech nonprofits.

Featured on Community Kubernetes Course Learn Python 3 Machine Learning in Python Getting started with Go Intro to Kubernetes

DigitalOcean Products Virtual Machines Managed Databases Managed Kubernetes Block Storage Object Storage Marketplace VPC Load Balancers

Welcome to the developer cloud

DigitalOcean makes it simple to launch in the cloud and scale up as you grow – whether you're running one virtual machine or ten thousand.

Learn More