Setting up a new project can be a time consuming endeavour. The world of front-end web development these days is much more complex than it used to be and there are now a whole load of tools and workflows to choose from to create and manage the development process.

In this video we'll discuss some of the features of a front-end development workflow and then a command line tool which can help you get set up and running FAST.

Front End Workflow

What's involved in setting up and working on a new project?

It used to be that all you needed was a browser, a text editor, an HTML file, a CSS file, jQuery and a handful of plugins.

Now we have package managers like npm and Bower, task runners like Grunt and Gulp, bundlers like Webpack or Browserify, local servers, automatic reloading, pre-processing, post-processing and all sorts of other things to help make your workflow faster and more efficient.

While all these things certainly do take away a lot of manual tasks and enable you to automate a large part of your development process, setting them up can be time consuming at best and daunting to the point of not even starting at worse.

If you want to see exactly how to set these things up step by step, we recently released a course on SitePoint Premium all about responsive web development that uses all these tools. I highly recommend checking it out.

However, if you're already on board with the benefits of these tools but cringe at the start of every new project at the prospect of setting them all up, I have an alternative to show you.

Yeoman

Yeoman is a command line tool that can help you automate the set up of new projects. It was developed by a number of very smart people and

helps you generate boilerplate code for modern websites and front-end web applications.

Yeoman is built on top of Node.js and is installed via npm so to follow along with this screencast or to set it up on your own machine you'll need to ensure you have Node installed.

With Node installed, open your command line tool of choice and install Yeoman with this single line command:

```
npm install -g yo
```

This will install the Yeoman command line tool globally (that's what the -g flag means) which you can use by running the yo command.

When you run this for the first time you'll see a command line menu with a handful of options:

- Install a generator
- Find some help
- or Get me out of here!

You can navigate this menu using your cursor keys and just kit enter to make your choice.

The Yeoman tool isn't particularly useful on its own but when combined with other tools, called Yeoman Generators, things start to get interesting.

A Yeoman Generator is an npm package that provides a configurable set of templates for creating a new project. You can search the web and discover a whole load of different generators for kick-starting all kinds of different web projects: angular apps, backbone apps, ember apps, react apps, boostrap sites, foundation sites, foundation emails, wordpress sites and thousands more.

If you're feeling particularly adventurous you can even create your own.

If you head to the Discovering Generators page on the Yeoman website you can search for what ever kind of project you want to create.

Since this is a Sass series, let's find something Sassy.

There's a great generator called Front end Sass which will generate a base project using Sass, Grunt, BrowserSync, code linting, image spriting, code minification and automatically add vendor prefixes with Autoprefixer. More details can be found on the generator's Github page.

To install the generator we need to install it as a global npm package.

```
npm install -g generator-front-end-sass
```

With the generator installed we can now create ourselves a project folder and then change directory into the newly created folder.

```
mkdir yeoman-sass
cd yeoman-sass
```

Within this project folder we can now run the Yeoman generator to scaffold out all the necessary files and folders for a Sass project complete with Gulp build tools and Autoprefixer.

```
yo front-end-sass
```

Eventually the generator will finish and we can take a look at what it did.

Here's the project open in Sublime Text. We started with an empty folder but after running the generator there is a whole lot of stuff in here.

Opening up the root project folder we have a development folder for all of our code and a node_modules folder for all the npm packages that the generator has installed. There are also a number of configuration files (sometimes called dotfiles) that are used to configure a number of the automated tasks that our newly generated project can perform to speed up our workflow.

In the development folder we have a js folder - for our scripts - and a sass folder for our sass.

The generator has automatically generated a solid folder structure for a well organised Sass project: a `styles.scss` file which will be compiled is made up of a number of partials from the `base` folder. These partials contain only a few lines of Sass rather than being overly prescriptive and forcing you to go down a particular path. There are some useful mixins for responsive design and even a `_shame.scss` for any quick and dirty fixes you want to clean up at a later date.

I find this generator to be a really good balance of giving just enough for a starting point without too much code that you might never use or even understand.

Not only does this generator provide a good starting point for organising your project, it comes with a load of tasks that can be run with Grunt via the command line to speed up and automate your development workflow. Let's have a look in the Gruntfile to see what it can do.

Grunt Tasks

As we saw in a previous episode about Compiling Sass with Grunt and Gulp, Grunt is a task runner that can automate common tasks that need to be run as part of the web development process.

These tasks are configured in a JavaScript file called the `gruntfile.js`. In this file we first load the grunt plugins and set up some configuration variables that tell grunt where to find our source code - the development folder - and where to output any compiled files

- to a dist folder.

We have a browserSync task which launches a local development server and watches files for changes (every time you save) and automatically injects new styles or reloads the page if the HTML or JS is updated.

We have a Sass task which compiles the Sass to CSS. It also generates source maps which we've also covered in a previous episode so check that out for more information on how useful source maps are.

Next we have a postCSS task which runs after the Sass has been compiled to CSS and in this case runs three post processors: adding pixel fallbacks for any rem values, automatically adding vendor prefixes based on caniuse.com data and then minifying the resulting CSS.
The result of running these will be saved to the `development/css` folder.

Next we have a handful of tasks that are used to check your code for errors and best practices - called linters - and tasks that generate documentation from comment blocks.

We have a task that copies files from the development folder to the destination folder which is what would be deployed as the working version of your project.

There are also tasks for compressing images and generating image sprites which can help reduce the page weight and speed up your site.

Finally we have a watch task that watches all the source files for changes and runs the appropriate Grunt task. For example, the `watch:styles` task will watch for changes in your Sass files and run the `sass` and `postcss` tasks.

After all the Grunt tasks have been configured, we register a series of tasks at the bottom of the file.

The first - and most useful - task is `grunt serve` which will launch browserSync, compile Sass and then watch your files for changes. There are other tasks registered for testing your code, generating docs or running all tasks in one go.

Phew, that was a lot to take in. So, how do we use these tasks? Let's demonstrate how we'd use this set up in development mode.

Back in the command line, make sure you're in your project folder and run `grunt serve`. This will run the "serve" task we were just looking at which will launch the development server, compile Sass and then start watching files for changes.

Your browser should automatically open but if not, you can browse to `localhost:3000` and see your work.

I'll open the browser and the code side by side to demonstrate how all these tasks work.

In the Sass let's add to the base styles. I'll create a variable and set the background color of the body. As soon as I save, Grunt kicks in and compiles the Sass and automatically injects the new styles to the page without reloading.

If I use CSS that needs vendor prefixes, like flexbox, these will automatically be added to the compiled styles via Autoprefixer.

This kind of automated development workflow is hugely beneficial. It can really speed up your work and save you doing a lot of manual repetitive tasks.

There's a lot more you can do with these kinds of tools - much more than we can demonstrate in a 5-10 minute video but hopefully this has demonstrated the kinds of things that are possible.