Feature detection and vendor prefixes in CSS

CSS has had to deal with vendor prefxies from very early on and some of those prefixes are still in use today. This post will look at how to use prefixes and how to automate the process, and how to use `@supports` to test if a feature is supported.

Adding CSS prefixes

That we're still dealing with vendor prefixes shows how long some of them have lasted and provide an object lesson on how to handle features in development

Developers didn't think of vendor prefixes as a way to preview features, they shipped them in production, something the CSS Working Group considers harmful.

Once you add a flag or vendor prefix and developers start use it in production, it becomes harder for browsers to change the feature works or its syntax. With flags, your users must actively enable it.

The standard way

The easiest way to handle different vendor prefixes is to just list them all. Browsers will ignore any rule they don't understand and if they understand more than one rule in a selector, the last one in document order wins; that's why we put the unprefixed version last.

```
.myClass {
  -webkit-transition: all 1s linear;
  -moz-transition: all 1s linear;
  -ms-transition: all 1s linear;
  transition: all 1s linear;
}
```

Although most vendor prefixes have been consolidated we should still be careful which ones we use and in what order:

- Microsoft and Opera adopted Blink for their browsers so all the prefixed

rules in Blink apply to Edge and Opera too
- IE 11 may stay with us a while longer so we still need the `-ms-` prefix for some properties
- Blink still uses `-webkit-` prefixed properties from before the fork

Automating the process

If we're willing to put our CSS through a build process or use technologies like SASS/SCSS or LESS that already require a compilation step we can also automate vendor prefixes with tools like Autoprefixer either as a standalone tool or with your favorite task runner.

This example uses `gulp-autoprefixer` and `gulp-sourcemaps` to automatically add prefixes to all the CSS files under the `src`.

```
const gulp = require('gulp');
const sourcemaps = require('gulp-sourcemaps');
const a'gulp-sourcemaps'uire('gulp-autoprefixer');

exports.default = () => (
  gulp.src('src'gulp-autoprefixer'/**/*.css')
    .pipe(sourcemaps.init())
    .pipe(autoprefixer())
    .pipe(sourcemaps.write('.'))
    .pipe(gulp.dest('dist'))
);
```

By default, Autoprefixer uses data from caniuse.com to determine what prefixes to use for the default set of browsers.

Test browser support for a feature using `@supports`

CSS way to test if a browser supports a given feature is to use `@supports` to query if a browser supports a feature.

```
@supports (display: grid) {
  .container {
    display: grid
  }
}
```

This example checks if the browser supports CSS grid (the `display: grid` rule) and uses it if it's supported. If the browser doesnt understand `@support` or `display: grid` it will ignore the full `@supports` block altogether.

> For more information on `@supports` see MDN's @supports article