65 PAGES OF PRACTICAL WEB DESIGN TIPS AND TECHNIQUES!

/develop/discover/design

/ issue 246 / october 2013 / www.netmagazine.com

net

issue 246

*Create slick CSS layouts

Build amazing user interfaces with the revolutionary Flexbox

MASTER • THE GUILD OF • MASTER DEVELOPERS •

MASTER BROWSER DEV TOOLS

We reveal how to get the most out of Firefox, IE, Opera and Safari. ***Plus*** *speed sites up in Chrome*

>Responsive wireframes
Techniques for HTML prototyping

>Develop with Grunt
Give your workflow a makeover

>Marketing analytics tips
How to measure social activity & more

Future

/contributors

We've scoured the web to find the best writers, designers and net gurus. From authors and activists to user experience specialists, we've got 'em all

Lauri Hynynen

Lauri (@laurihy) is a developer at Backlift. Read his tutorial on page 100 on building a form by using Backlift to create a Facebook application just by editing some files in Dropbox.

http://backlift.com

Stephen Hay

Stephen (@stephenhay) is a frontend design and development consultant. Read his tutorial on page 74 about the new CSS spec, The Flexible Box Layout Module, aka Flexbox.

www.the-haystack.com

Andi Smith

Andi (@AndiSmith) is a presentation technical architect at ideas and innovation agency AKQA. He provides valid frontend solutions for large-scale digital solutions and also works with the latest HTML5 and JavaScript APIs. He's worked alongside clients such as Heineken, Nike, Unilever and MINI. Turn to page 34 for his secrets for optimising your development workflow using browser tools.

www.andismith.com

Jared Ponchot

Jared (@jponch) is design director at interactive strategy, design and development company Lullabot. Turn to page 80 for his tutorial on using the Foundation framework to translate a sketch to a responsive HTML prototype.

www.lullabot.com

Dennis Odell

Dennis (@denodell) is head of web development at ideas and innovation company AKQA. Turn to page 94 where he shows you how to simplify day-to-day frontend development tasks and streamline your workflow by automating repetitive tasks with Grunt, the JavaScript task runner.

www.akqa.com

Addy Osmani

Addy (@addyosmani) works on the Chrome team at Google. Turn to page 46 for how to use Chrome DevTools to speed up your site.

http://addyosmani.com-

/develop/discover/design

.net

Future Publishing, 30 Monmouth Street, Bath BA1 2BW
Telephone +44 (0)1225 442244 **Website** www.netmagazine.com

Editorial
Editor Oliver Lindberg, oliver.lindberg@futurenet.com
Deputy editor Martin Cooper, martin.cooper@futurenet.com
Production editor Jenny Williams, jenny.williams@futurenet.com
Digital editor Tanya Combrinck, tanya.combrinck@futurenet.com
Art editor Mike Brennan, mike.brennan@futurenet.com

Contributors
Matt Aebersold, Aral Balkan, Romain Bourdieux, Ross Bruniges, Ruth Burr, Mike Buzzard, Kevin Charlton, Brad Colbow, Ryan Crawford, Anna Dahlström, Ian Devlin, Tom Dougherty, Ryan Downie, Ben Dyer, Frank Fenten, Jack Franklin, Adam Gatenby, Matt Gifford, Craig Grannell, Susan Hallam, Stephen Hay, Whitney Hess, Lauri Hynynen, Jim Kalbach, Laura Kalbag, Mark Kirby, A.J. Kohn, Mustafa Kurtuldu, Karen Lewis, Elliot Lings, Michael Litman, Jenn Lukas, Damon Mangos, Gary Marshall, Bryson Meunier, Lorna Mitchell, Katie Moffat, Jordan Moore, Nick Moules, Luke Murphy-Wearmouth, James Oconnell, Dennis Odell, Addy Osmani, Thomas Pomarelle, Jared Ponchot, Robert Reinhard, Harry Roberts, Alex Russell, Luke Scanlon, Fadi Shuman, Chris Skelton, Jonathan Smiley, Andi Smith, Graham Smith, Brian Suda, John Taylor, Jim Thacker, Alex Turner, Alistair Tweedie, Rob Walker, Trent Walton, Sam Williams, Paul Wyatt

Photography Kristinn Magnússon
Group Art Director Steve Gotobed
Creative Director Robin Abbott
Editorial Director Jim Douglas

Advertising
Advertising Director Charlie Said, 020 7042 4142, charlie.said@futurenet.com
Account Manager Julian Tozer, 020 7042 4273, julian.tozer@futurenet.com
Account Manager Ross Arthurs, 020 7042 4128, ross.arthurs@futurenet.com
Account Manager Laura Watson, 020 7042 4122, laura.watson@futurenet.com
London Sales Director James Ranson, 020 7042 4163, james.ranson@futurenet.com

Marketing
Group Marketing Manager Philippa Newman, philippa.newman@futurenet.com

Circulation
Trade Marketing Director Daniel Foley, daniel.foley@futurenet.com

Print & Production
Production Coordinator Roberta Lealand, roberta.lealand@futurenet.com

Licensing
Licensing & Syndication Director Regina Erak, regina.erak@futurenet.com
Tel + 44 (0)1225 732359

Future Publishing Limited
Editor in chief Dan Oliver, dan.oliver@futurenet.com
Head of creative and design Declan Gough, declan.gough@futurenet.com
Managing director Nial Ferguson, nial.ferguson@futurenet.com
Chief Executive Officer (UK) Mark Wood, mark.wood@futurenet.com

Subscriptions
Phone our UK hotline 0844 848 2852; **international** (+44) (0) 1604 251045
Subscribe to .net magazine online at www.myfavouritemagazines.co.uk

NEXT ISSUE ON SALE: 8 October 2013

Printed in the UK by Philtone Litho Ltd.
Distributed in the UK by Seymour Distribution Ltd,
2 East Poultry Avenue, London EC1A 9PT. Tel: 0207 429 4000

abc
Print 12,703
Digital 3,606
A member of the Audited Bureau of Circulations

The ABC combined print, digital and digital publications circulation for Jan–Dec 2012 is
16,309

Future

Future produces high-quality multimedia products which reach our audiences online, on mobile and in print. Future attracts over 50 million consumers to its brands every month across five core sectors: Technology, Entertainment, Music, Creative and Sports & Auto. We export and license our publications.

Future plc is a public company quoted on the London Stock Exchange (symbol: FUTR).
www.futureplc.com

Chief executive Mark Wood
Non-executive chairman Peter Allen
Chief financial officer Graham Harding
Tel +44 (0)207 042 4000 (London)
Tel +44 (0)1225 442 244 (Bath)

THE GUILD OF • MASTER DEVELOPERS •

MASTER BROWSER DEV TOOLS

Inside every browser are tools to aid developers. **Andi Smith** shares his tricks and secrets for optimising your development workflow

Once upon a time, debugging a web page involved inserting copious amounts of alerts into our JavaScript, hitting refresh, and then hammering every alert box OK button until we found the information we wanted. Thankfully, over the years, debugging a website has got easier, with the main browsers making specialist, built-in tools available to developers.

Browser developer tools allow us to inspect, edit, debug, log and profile our HTML, CSS and JavaScript, and include an exhaustive range of features and functionality to aid us in these tasks. You can improve your development workflow by getting to know how these tools work in every browser. Below are 20 tips, split into three categories (beginner, intermediate and advanced) to help you get started.

For beginners

01 Log multiple console values in different formats

If we use string concatenation in our logs when debugging objects and arrays, the console will return the string value of an object. Instead, log multiple values from **console.log** by supplying multiple arguments:

```
console.log('Object output: ' + myObject);
// returns Object output: [object Object]
console.log('Object output:', myObject);
// returns Object output: Object {key: "value",
key2: "value"}
```

As well as the standard log, there are other types of logging available, which can help you to distinguish between the different types of console information:

```
console.log('Captain\'s Log 1', new Date());
console.info('Ammo supplies moderate');
console.debug('Shield\'s are up');
console.warn('Unidentified craft approaching');
console.error('Fire in the cargo bay');
```

You can preserve logs between pages in Chrome and Opera 15 by right-clicking on the console output and selecting **Preserve Log Upon Navigation**. In Internet Explorer (IE), this option is available in the **Tools** menu bar, under **Clear Entries On Navigate**.

In Firebug (http://getfirebug.com), the **Persist** option above the console will keep the console history. Currently, Firefox preserves console logs by default. In a future version, this option will be in the **Settings** menu.

02 Use the console in any panel

You can run an instance of the console from any of the other panels in Chrome, IE11, Opera 15 and Safari.In Chrome and Opera 15, press the **Esc** key. In IE11, the keyboard shortcut is **Ctrl + '**. Safari has a small console prompt available below the main panel by default.

Combining the console alongside tabs such as the DOM Inspector and Scripts panel can provide a very powerful way to inspect what's happening within your DOM and scripts.

03 View your site as responsive

There are various options available for testing responsive pages in the

>>

Words Andi Smith is a presentation technical architect at ideas and innovation agency AKQA. He provides valid frontend solutions for large-scale digital solutions and also works with the latest HTML5 and JavaScript APIs. He's worked alongside clients such as Heineken, Nike, Unilever and MINI www.andismith.com

Image Mike Brennan is art editor of .net www.twitter.com/mike_brennan01

Using the browser dev tools

Browser developer tools provide a powerful platform for examining our web pages. Here's a quick rundown of how to do so:

Accessing the browser dev tools

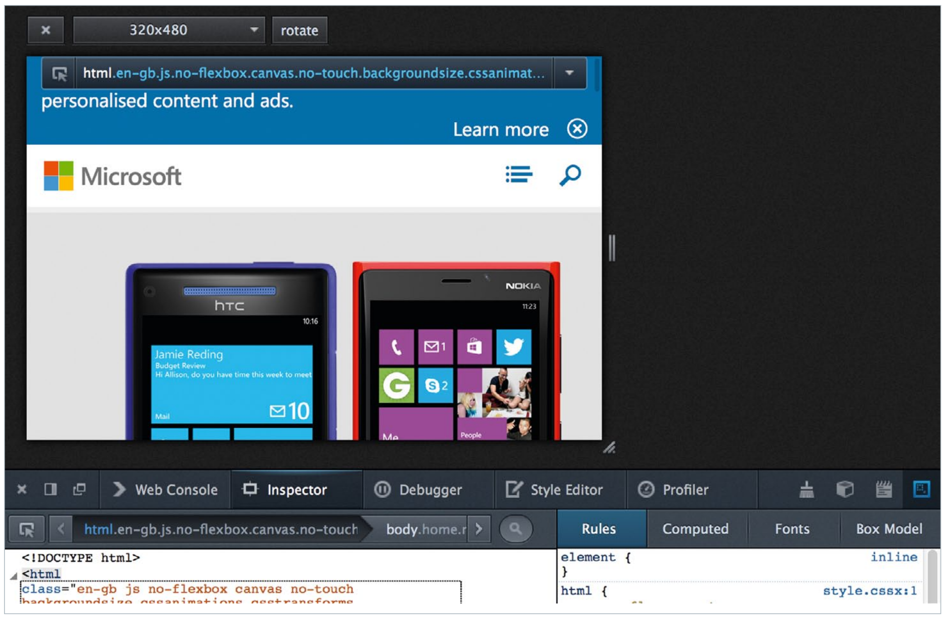
Each browser provides a menu shortcut for accessing the developer tools, plus keyboard shortcuts. In most browsers, you can access the developer tools by pressing **F12** or **Ctrl**, **Shift** and **I** on Windows and Linux or **Cmd**, **Opt** and **I** on Mac.

In all browsers (except IE) you can also open the tools by right clicking on the page and selecting Inspect Element. The tools will automatically highlight the element you have selected on the page. In Safari, enable the developer tools first. Do this by checking the option in **Preferences > Advanced > Show Develop Menu in menu bar**.

Base functionality

All browsers have similar base functionality:

- **Console:** displays logs, warning and errors; a JavaScript console writes scripts to interact with our page.
- **Elements/Inspector/HTML:** shows the browser's interpretation of the HTML code of current pages. Nodes and attributes on the pages can then be easily manipulated directly from this DOM inspector for real-time editing.
- **Styles:** lists all styles from our stylesheet applied to the element currently selected. Also contains a 'computed' view showing all the styles the browser is applying to our element, and a box model view of the element showing margin, border, padding, width and height. The CSS Styles panel also allows for real-time editing.
- **Script/Sources/Debugger:** a script tab for viewing and debugging JavaScript running on our page. We can set breakpoints in our code and step through line by line to check it is functioning as expected.
- **Profiler:** this captures and analyses JavaScript performance.
- **Network:** a traffic timeline for viewing resources loaded on to the page. These are listed with HTTP status, file size, header information and response times.



320x480
rotate
html.en-gb.js.no-flexbox.canvas.no-touch.backgroundsize.cssanimat...
personalised content and ads.
Learn more
Microsoft
Web Console
Inspector
Debugger
Style Editor
Profiler
html.en-gb.js.no-flexbox.canvas.no-touch
body.home.r
Rules
Computed
Fonts
Box Model

```
<!DOCTYPE html>
<html
class="en-gb js no-flexbox canvas no-touch
```

```
element {                inline
}
html {              style.cssx:1
```

Responsive mode Chrome, Firefox, IE11 and Opera 15 all offer dedicated responsive modes

>> browser in addition to the traditional method of simply resizing the browser.

In Chrome, Firefox, Opera 15 and Safari 6.1, docking the tools to the right enables users to achieve finer control over resizing their content, while also facilitating full use of the tools.

In Chrome and Opera 15, the option to dock to right is found to the bottom left of the tools: clicking and holding the **Dock** button will enable you to select between three docking options. In Firefox, the **Dock To Right** button is in the top-left of the toolbar, next to the **Close** button; in Safari 6.1, it's located in the top right-hand corner and is only available once you've undocked the tools.

Chrome, Firefox, IE11 and Opera 15 also include dedicated responsive modes, which can be used to set the browser window to a defined width and height. This has the advantage of displaying how much content will be shown in the default viewport area.

In Chrome and Opera 15 developer tools, open the **Settings** menu (the cog found in the bottom right) and select the **Overrides** tab. In this tab is an option called **Device metrics** that allows us to specify our own width and height. Alternatively, if you tick the **User Agent** option and select a device, it will automatically fill in the correct dimensions. You can then untick the **User Agent** option if you don't require it. In Firefox, the **Responsive Design Mode** is the top-right icon in the menu. Choosing this option presents a smaller viewport with a drop-down to choose different device dimensions. In IE11, this mode can be found in the Emulation tab.

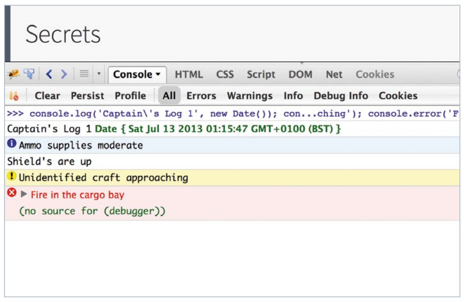
There are various options available for testing responsive pages in the browser, in addition to simply resizing it

04 View available CSS properties

In Chrome, IE11, Opera 15 and Safari, you can view the available CSS properties on an attribute by pressing **Ctrl** and **Space** when the CSS attribute field is blank.

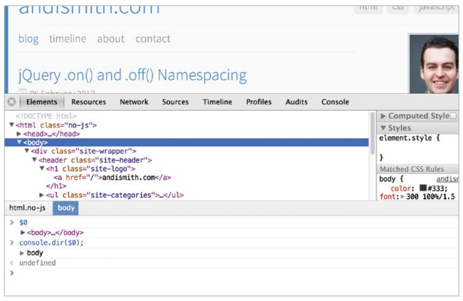
05 Reference DOM elements from the console

While working in the console, we may need to reference an element from the DOM. There are various shortcuts to select elements quickly.



```
>>> console.log('Captain\'s Log 1', new Date()); con...ching'); console.error('F
Captain's Log 1 Date { Sat Jul 13 2013 01:15:47 GMT+0100 (BST) }
Ammo supplies moderate
Shield's are up
Unidentified craft approaching
▶ Fire in the cargo bay
  (no source for (debugger))
```

Console tip Get to know the different types of logging

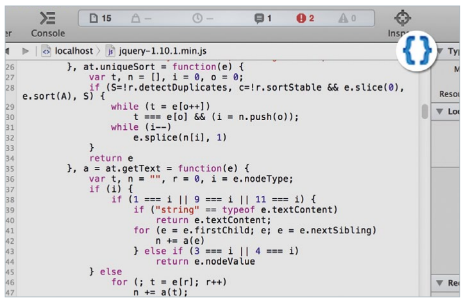


```
<!DOCTYPE html>
▼<html class="no-js">
  ▶<head>…</head>
  ▼<body>
    ▼<div class="site-wrapper">
      ▼<header class="site-header">
        ▼<h1 class="site-logo">
            <a href="/">andismith.com</a>
          </h1>
        ▶<ul class="site-categories">…</ul>
```

```
element.style {
}
body {
   color: #333;
font: ▶300 100%/1.5
```

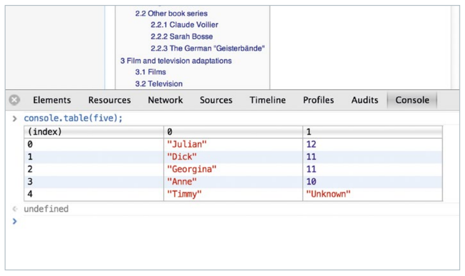
```
> $0
  ▶<body>…</body>
> console.dir($0);
  ▶ body
< undefined
>
```

Use the console anywhere Open a console in any panel



```
        }, at.uniqueSort = function(e) {
            var t, n = [], i = 0, o = 0;
            if (S=!r.detectDuplicates, c=!r.sortStable && e.slice(0),
e.sort(A), S) {
                while (t = e[o++])
                    t === e[o] && (i = n.push(o));
                while (i--)
                    e.splice(n[i], 1)
            }
            return e
        }, a = at.getText = function(e) {
            var t, n = "", r = 0, i = e.nodeType;
            if (i) {
                if (1 === i || 9 === i || 11 === i) {
                    if ("string" == typeof e.textContent)
                        return e.textContent;
                    for (e = e.firstChild; e; e = e.nextSibling)
                        n += a(e)
                } else if (3 === i || 4 === i)
                    return e.nodeValue
            } else
                for (; t = e[r]; r++)
                    n += a(t);
```

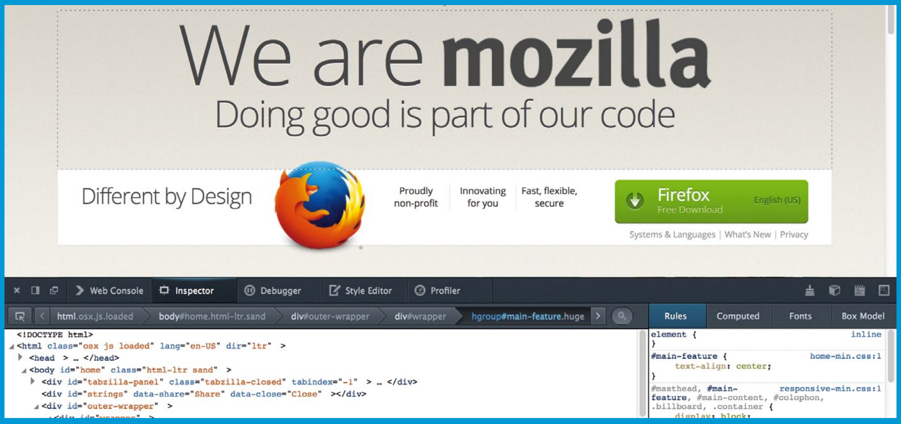
Pretty print Look for **{}** icon to un-minify JavaScript code



```
> console.table(five);
  (index)        0             1
  0              "Julian"      12
  1              "Dick"        11
  2              "Georgina"    11
  3              "Anne"        10
  4              "Timmy"       "Unknown"
< undefined
>
```

Tabular logging Use **console.table** to log arrays or objects

Current state of browser developer tools



```
<!DOCTYPE html>
<html class="osx js loaded" lang="en-US" dir="ltr" >
  <head > … </head>
  <body id="home" class="html-ltr sand" >
    <div id="tabzilla-panel" class="tabzilla-closed" tabindex="-1" > … </div>
    <div id="strings" data-share="Share" data-close="Close" ></div>
    <div id="outer-wrapper" >
```

```
element {                                inline
}
#main-feature {                   home-min.css:1
    text-align: center;
}
#masthead, #main-          responsive-min.css:1
feature, #main-content, #colophon,
.billboard, .container {
    display: block;
```

Catching up The dev tools in Firefox receive regular updates and are catching up to the tools in other browsers

Frequent release cycles mean we receive new versions of our browsers regularly, which means we also receive frequent updates to our developer tools.

- **Chrome** developer tools have had the fullest feature set for some time now. Each new release tends to add more new and exciting features such as workflows, snippets, canvas inspection and source-mapping support.

- **Firebug** was once the preferred choice for browser tooling, but it has recently fallen behind. Firebug updates are still frequent – often every two or three weeks – but they tend to be incremental changes rather than the larger feature changes of other browsers.

- **Firefox** has had its own developer toolbar for some time now. As a late arrival on the scene, it's taken a while for Mozilla's Firefox to catch up with its competitors. However, significant updates have been provided with each of the new releases.

- **Internet Explorer** developer tools have been at risk of stagnation between recent releases, with very little changing between IE9 and IE10. However, the tools have been given an overhaul for IE11 with a refreshed display and many new features. (Also see 'New IE11 dev tools' on page 38 for some more information on new features).

- **Opera** developer tools (known as Dragonfly) have always been very competent at debugging websites – but with the release of Opera 15 and the move to Blink, these tools are now identical to Chrome's. It's likely, though, that Dragonfly will return in one form or another in a later version.

- **Safari** revamped its developer tools with the release of Safari 6. But a combination of the UI overhaul and the change of most options to icons made it difficult for developers to understand. The latest Safari 6.1 beta, however, shows great strides are being made to create a much clearer interface.

We can retrieve a single element by using **$**. For example, if our element had an ID of **magazine**, we would type **$('#magazine')**. If it had a class of **latest**, we would type **$('.latest')**;.

Here, **$** is shorthand for **document.querySelector**. Therefore, it only returns the first match rather than an array of matches as in JavaScript libraries such as jQuery. If the **$** variable is already in use, then this functionality is overridden by the library using it.

In some older browser developer tools and in IE11, **$** will retrieve an ID, while **$$** is used to retrieve a class. In these cases, the hash or dot are not required. For example:

```
$('magazine');
$$('latest');
```

Tools that have the newer functionality will warn you if you use the old syntax.

06 Reference the current or previously selected DOM element

If you have an element currently selected in your DOM, you can use the reference **$0** to call it within your code. For example, to see the current element's inner HTML, you would type the following: **$0.innerHTML**.

In Chrome, Firebug 1.12, IE11, Opera and Safari, you can retrieve elements you have previously highlighted by using **$1 - $4**. In Firebug, you can use **$n(2) - $n(5)**.

07 Set conditional breakpoints

Rather than hitting a breakpoint whenever we reach a line of code, we can set a breakpoint to trigger based on whether a particular expression is true.

We can create a conditional breakpoint within our script panel by creating a normal breakpoint, right-clicking and selecting **Edit, Condition...** or **Add conditional breakpoint** (depending on the browser). This can be especially useful for breakpoints set within loops. For instance, by adding a conditional expression of **i = 10**, we'll hit the breakpoint only when this expression is true.

08 Debug minified code

Placing breakpoints on JavaScript makes debugging much easier, but if your code has already made it to production then it may have already been minified. How can you debug minified code? Helpfully, some of the browsers have an option to un-minify your JavaScript.

In Chrome and Opera 15, simply select the **Sources** tab, find the relevant script file and then press the **{ }** (Pretty Print) icon that's located within the bottom panel.

In IE, click the tool icon by the script selection drop-down to find the option to format the JavaScript. Older versions of Opera and Safari 6.1 will automatically un-minify your code.

Intermediate level

09 Use logging based on expressions

We can use **console.assert()** to provide conditional logging on our pages. Assert works by evaluating an expression, before logging a console message if the evaluated expression is false. If the expression is true on the other hand, no message is logged.

For example:

```
var i = 11;
console.assert(i < 10, 'Unexpected value for i: ' + i);
// returns Assertion failed: Unexpected value for i: 11
```

At the time of writing, **assert** is supported in all browser developer tools (including Firebug) except Firefox Developer Tools.

10 List the properties of an object

Using **console.dir(obj)** produces a list of properties available on an object. The properties are interactive; they can be altered on the fly. This feature is especially useful with DOM objects (for example, document), as traditional console log commands show the HTML output rather than a list of object properties.

>>

>> 11 Profile JavaScript from the console

We can profile parts of our JavaScript to find performance bottlenecks directly from the console via two different approaches.

By using **console.time**, we can perform a quick analysis of the time taken to run our code, while **console.profile** provides a greater breakdown by showing how long it took the individual functions to run.

To use the timer, we need to provide a label as a parameter to **console.time**. The label allows us to run multiple timers concurrently:

```
console.time('timer 1');
```

To end the timer, we provide the same label to **console.timeEnd**:

```
console.timeEnd('timer 1');
```

At this point, the console will log the number of milliseconds the timer had been running for.

console.profile works in the same way, with a corresponding **console.profileEnd** method. However, the report produced after running it is much more detailed. Depending on the browser developer tool, the profile will either be logged out to the console or, in the case of Chrome and Opera 15, to the **Profiles** tab.

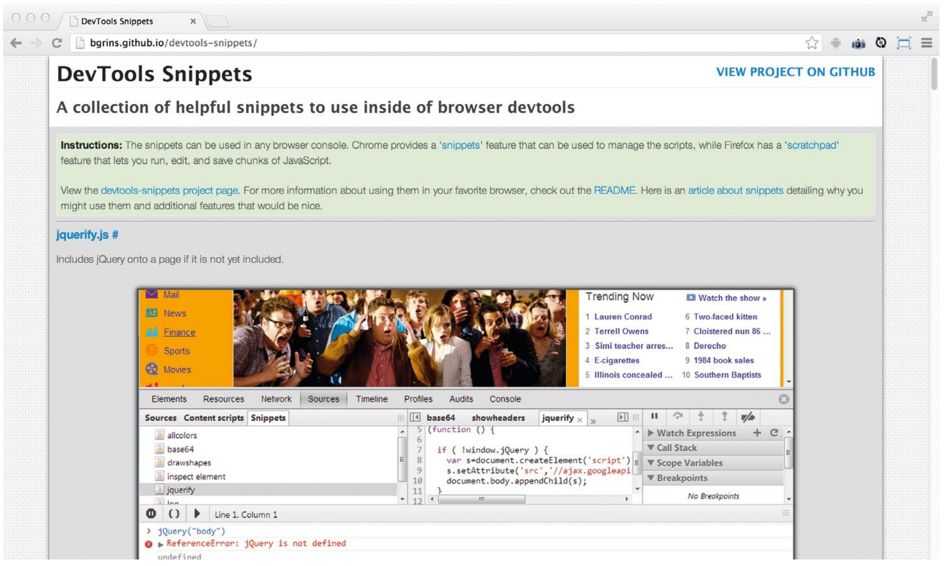
12 Log data in a table

If the data we are logging to the console is a multidimensional array, or an object, we can use **console.table** in Chrome, Firebug and Opera 15 to output the data into a table we can sort.

To log a multidimensional array as a table:

```
var five = [["Julian", 12], ["Dick", 11], ["Georgina",
11], ["Anne", 10], ["Timmy", "Unknown"]];
console.table(five);
```

Logging an object as a table works in the same way, except that property names can be used for column and row names. Alternatively, we can add a second argument to specify which columns to output. For example, see the following:



DevTools Snippets

bgrins.github.io/devtools-snippets/

DevTools Snippets

VIEW PROJECT ON GITHUB

A collection of helpful snippets to use inside of browser devtools

Instructions: The snippets can be used in any browser console. Chrome provides a 'snippets' feature that can be used to manage the scripts, while Firefox has a 'scratchpad' feature that lets you run, edit, and save chunks of JavaScript.

View the devtools-snippets project page. For more information about using them in your favorite browser, check out the README. Here is an article about snippets detailing why you might use them and additional features that would be nice.

jquerify.js #

Includes jQuery onto a page if it is not yet included.

```
(function () {

  if ( !window.jQuery ) {
    var s=document.createElement('script')
    s.setAttribute('src','//ajax.googleapi
    document.body.appendChild(s);
  }
```

```
> jQuery("body")
✖ ▸ ReferenceError: jQuery is not defined
undefined
```

DevTools Snippets Brian Grinstead is maintaining a list of useful snippets to aid debugging (http://netm.ag/snippets-246)

```
var devtools = {"chrome": { "creator": "Google",
  "version": 28 }, "firefox": { "creator": "Mozilla",
  "version": 22 }, "internet-explorer": { "creator":
  "Microsoft", "version": 11,"plugins":"Silverlight"
  }};
console.table(devtools, ['creator','version']);
```

If a property is missing from an object, the table will display undefined for this table cell.

13 Run snippets of code

Sometimes when debugging we want to run the same snippet of code continuously against our page; at other times it would be handy to include a saved snippet to help us debug our code. We can run any JavaScript code from the console already, but both Chrome, Opera 15 and Firefox also enable us to save these snippets so that we can use them again and again.

You can find **Snippets** in the **Sources** tab of Chrome and Opera 15 dev tools, in the left-side menu after **Sources** and **Content Scripts**. Here you can create new snippets and run previously created snippets, provided as a list accessible by all your sites. In Firefox you'll need to use Scratchpad (https://blog.mozilla.org/devtools/2011/08/15/introducing-scratchpad), which allows us to load, save and run JavaScript snippets by making use of the more traditional file system method.

Find a list of useful snippets for tasks such as adding jQuery to a page and listing all colours at http://bgrins.github.io/devtools-snippets.

14 Emulate a touch device

Building a site for smartphones within a browser can be difficult, especially if you have to provide touch-specific functionality like swiping.

In Chrome and Opera 15, you can pretend to be a device that supports touch and emulate touch events directly within the browser. In order to do this, open the **Settings** menu, select the **Overrides** tab and then **Emulate touch events**.

In Chrome, this option will also replace the cursor with a circle. You can then tell not only which mode you are in, but also how large the surface area for a touch click should be based on Apple recommendations.

New IE 11 dev tools

IE11, available for Windows 7 and 8.x, offers a new set of developer tools.

Some of the new features in the transition from IE10 to IE11 include:

New DOM Inspector/style features:

- Improved highlighting of selected elements: elements selected in the DOM Inspector now have a blue overlay.
- New breadcrumb navigation shows the DOM path to the current element, allowing rapid navigation up and down the path hierarchy.
- You can now change the order of DOM elements by dragging and dropping.
- **Trace** is a new feature that provides a concise summary list of all the properties applied from your stylesheets.
- The ability to filter computed styles.
- A new **Events** panel that allows us to inspect events attached to DOM elements.

New console features:

- The **window.console** object is now defined even when developer tools are closed.
- Console can now be opened from any pane: IE11 uses the keyboard shortcut **Ctrl + '**.
- Support for **$** as a shortcut for **getElementById** and **$$** as a shortcut for **querySelector**.
- Objects within console can be inspected.
- Support for **console.count, console.group, console.trace** and **console.time**.

New scripting/debugging features:

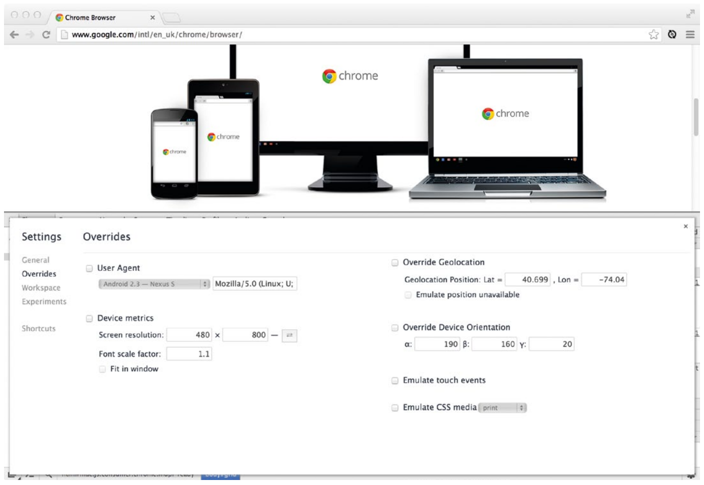
- The whole journey for debugging in IE has been improved. The **Start Debugging** button has been removed.
- Hover over variables to see values.
- Break on exceptions.
- You can **Set next statement** to ensure your debugger does not skip code wrapped in library functions.
- Run a script to see where the cursor is.
- Open multiple scripts simultaneously.

New performance tool:

- Called the **UI Responsiveness Tool**.
- It has a Timeline mode, similar to Chrome's.
- Shows a graph for actual frames per second (fps) against 60fps and 30fps.

New emulation features:

- Emulate for responsive web design.
- Emulate for Windows Phone.
- Fake device location via geolocation.


Settings — Overrides

- General
- Overrides
- Workspace
- Experiments
- Shortcuts

☐ User Agent: Android 2.3 — Nexus S | Mozilla/5.0 (Linux; U;

☐ Device metrics
Screen resolution: 480 × 800
Font scale factor: 1.1
☐ Fit in window

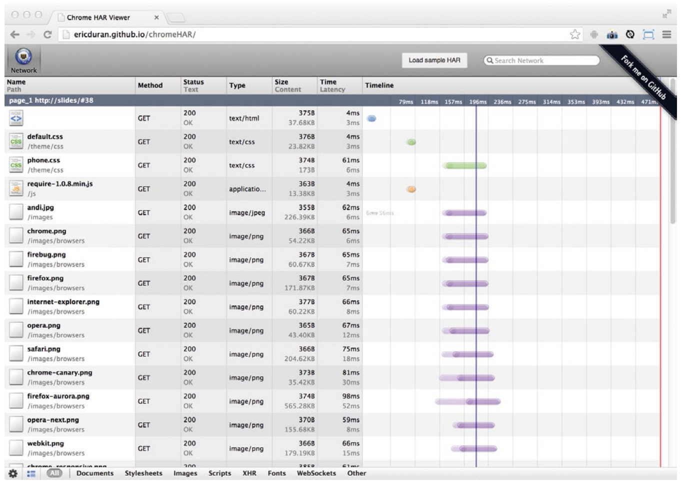
☐ Override Geolocation
Geolocation Position: Lat = 40.699 , Lon = -74.04
☐ Emulate position unavailable

☐ Override Device Orientation
α: 190 β: 160 γ: 20

☐ Emulate touch events

☐ Emulate CSS media: print

Emulation Chrome's emulation tools include geolocation, touch, user agents and print


Chrome HAR Viewer — ericduran.github.io/chromeHAR/

page_1 http://slides/#38

Name / Path	Method	Status / Text	Type	Size / Content	Time / Latency
	GET	200 OK	text/html	375B 37.68KB	4ms 3ms
default.css /theme/css	GET	200 OK	text/css	376B 23.82KB	4ms 3ms
phone.css /theme/css	GET	200 OK	text/css	374B 173B	61ms 6ms
require-1.0.8.min.js /js	GET	200 OK	applicatio...	363B 13.38KB	4ms 3ms
andi.jpg /images	GET	200 OK	image/jpeg	355B 226.39KB	62ms 6ms
chrome.png /images/browsers	GET	200 OK	image/png	366B 54.22KB	65ms 6ms
firebug.png /images/browsers	GET	200 OK	image/png	367B 60.67KB	65ms 7ms
firefox.png /images/browsers	GET	200 OK	image/png	367B 171.87KB	65ms 7ms
internet-explorer.png /images/browsers	GET	200 OK	image/png	377B 60.22KB	66ms 8ms
opera.png /images/browsers	GET	200 OK	image/png	365B 43.40KB	67ms 12ms
safari.png /images/browsers	GET	200 OK	image/png	366B 204.62KB	75ms 18ms
chrome-canary.png /images/browsers	GET	200 OK	image/png	373B 35.42KB	81ms 30ms
firefox-aurora.png /images/browsers	GET	200 OK	image/png	374B 565.28KB	98ms 52ms
opera-next.png /images/browsers	GET	200 OK	image/png	370B 155.68KB	59ms 8ms
webkit.png /images/browsers	GET	200 OK	image/png	366B 179.19KB	66ms 15ms

Network
Load sample HAR
Search Network
Fork me on GitHub
Timeline
79ms 118ms 157ms 196ms 236ms 275ms 314ms 353ms 393ms 432ms 471ms
All Documents Stylesheets Images Scripts XHR Fonts WebSockets Other

Network traffic See another user's network traffic for your site with Chrome HAR

15 Emulate a print stylesheet

While checking styles for print in the past, we would have to rely on **Print Preview** or **Print to PDF**. Recent versions of Chrome allow us to emulate the media type so we can see how print (or any of the other media types) looks directly in our browser. To do this, open the Chrome Developer Tools **Settings** menu and choose the **Overrides** tab. At the bottom of the list of options is **Emulate CSS media**. Select the checkbox and choose **Print**.

Pre-processors such as Sass are popular development aids, but the output they generate can be difficult to debug

16 What have I changed?

If you've made a number of changes to your CSS or JavaScript within your developer tools, it can be difficult to keep track of what you've already changed. In the **Sources** tab of Chrome and Opera 15 developer tools you can find out what modifications have been made by opening the **Sources** side panel, right-clicking and selecting **Local Modifications**. A list of the changed files will be displayed, together with timestamps for when they were changed.

By expanding any one of these changes you are given a before/after comparison. You can also reverse any changes you are no longer happy with. By right-clicking on the file, we can save the new CSS or JavaScript file.

Advanced

17 Export network data

If you've ever had a situation where a page works fine on your own machine, but on a machine in another location an image is missing or a piece of content does not load correctly, then being able to view that machine's network traffic can help us to debug the problem. In Chrome and Opera 15, we can export the network timeline by right-clicking within the **Network** tab and selecting **Save as HAR** from the context menu. HAR stands for HTTP Archive, a format for storing network traffic from our site. You can duplicate the effect in other browsers by running a network monitoring program such as Fiddler (http://fiddler2.com) or Charles (www.charlesproxy.com) and exporting the HAR from there.

We can then attach this to a bug report and load this HAR file on another machine by using a HAR reader such as chromeHAR (http://ericduran.github.io/chromeHAR).

18 Use Sass source maps

Preprocessors such as Sass (http://sass-lang.com) are popular development aids, but the output they generate can be difficult to work with in the dev tools. For example, the line numbers in your style panel won't correspond to the ones in your Sass file. Debugging gets much harder if you use minification to remove unnecessary characters (such as comments and newlines) and uglification to optimise and shrink variable names.

Using source maps we can map our generated CSS and JavaScript files back to the source, giving us more control when debugging in developer tools. Sass 3.3 and above come with full support for source maps. First, check that you have version Sass 3.3 or above running by entering:

```
gem list sass
```

If the version is younger than 3.3.0.alpha, you can run an install with:

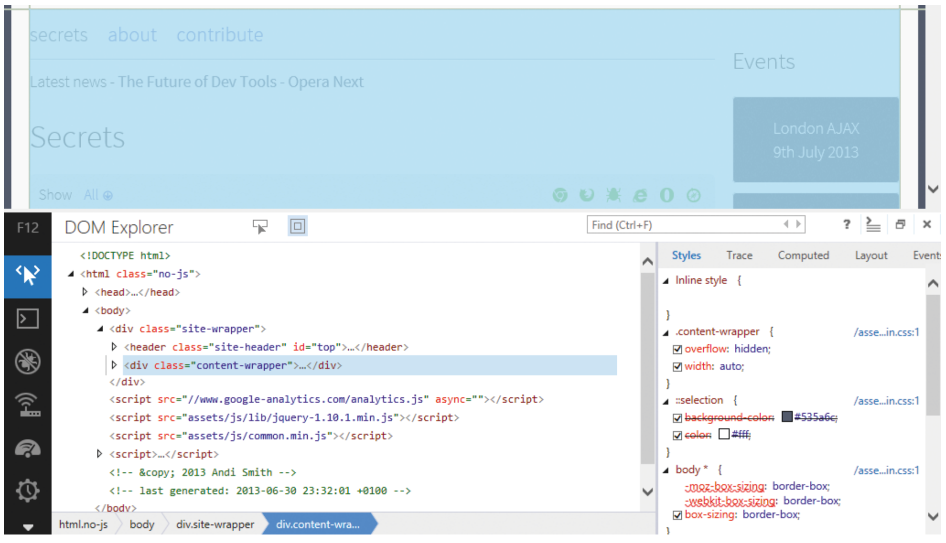
```
gem install sass --pre
```

Depending on how you compile your Sass files, there are various ways to create .map files. Two of the most popular are with the Sass **watch** command and with Grunt.

If you're using Sass **watch**, you need to add the **--sourcemap** parameter to your command:

```
sass --watch scss/common.scss:css/
common.min.css --style
compressed --sourcemap
```

>>


```
<!DOCTYPE html>
<html class="no-js">
  <head>…</head>
  <body>
    <div class="site-wrapper">
      <header class="site-header" id="top">…</header>
      <div class="content-wrapper">…</div>
    </div>
    <script src="//www.google-analytics.com/analytics.js" async=""></script>
    <script src="assets/js/lib/jquery-1.10.1.min.js"></script>
    <script src="assets/js/common.min.js"></script>
    <script>…</script>
    <!-- &copy; 2013 Andi Smith -->
    <!-- last generated: 2013-06-30 23:32:01 +0100 -->
  </body>
```

```
Inline style {
}
.content-wrapper {                 /asse...in.css:1
  overflow: hidden;
  width: auto;
}
::selection {                      /asse...in.css:1
  ~~background-color: #535a6c;~~
  ~~color: #fff;~~
}
body * {                           /asse...in.css:1
  ~~-moz-box-sizing~~: border-box;
  ~~-webkit-box-sizing~~: border-box;
  box-sizing: border-box;
}
```

New tools IE11 offers a whole new suite of developer tools

Latest Version

- Stable: v0.4.1
- Development: N/A

Optimizing Your Workflow with Grunt: a new class on air from the experts at Bocoup.

Ads by Bocoup.

Latest News

Grunt 0.4.1 released

Why use a task runner?

In one word: automation. The less work you have to do when performing repetitive tasks like minification, compilation, unit testing, linting, etc, the easier your job becomes. After you've configured it, a task runner can do most of that mundane work for you—and your team—with basically zero effort.

Why use Grunt?

The Grunt ecosystem is huge and it's growing every day. With literally hundreds of plugins to choose from, you can use Grunt to automate just about anything with a minimum of effort. If someone hasn't already built what you need, authoring and publishing your own Grunt plugin to npm is a breeze.

Grunt work Modules for Grunt such as grunt-contrib-sass and grunt-contrib-uglify can create source maps for you

>> If you're using the Grunt grunt-contrib-sass module, specifying the option **sourcemap: true** will compile source maps during build.

In Chrome, you need to enable the **Experiments** tab. If you haven't done so already, navigate to **chrome://flags** and choose **Enable Developer Tools experiments**. You'll need to press the **Relaunch Now** button at the bottom of the page to apply your changes.

Now go to the **Settings** panel of the developer tools, click the **General** tab and activate **Enable source maps**. Click the **Experiments** tab and check **Support For Sass**.

After restarting your browser, you will now be able to view line numbers of the Sass files directly.

19 Use JavaScript source maps

Like Sass source maps, the JavaScript equivalent provides a map back to the original formatted source. We can create a JavaScript source map using various different tools such as Google's Closure Compiler (https://developers.google.com/closure/compiler) or Grunt's Uglify (https://github.com/gruntjs/grunt-contrib-uglify) plug-in that builds .map files for you. RequireJS also has experimental support for source maps (http://requirejs.org/docs/optimization.html#sourcemaps).

When using Closure Compiler, we need to run the following :

```
java -jar compiler.jar –js common.js –create_
   source_map ./common.js.map –source_map_
   format=V3 –js_output_file common.min.js
```

With Closure Compiler, we also need to add a **sourceMappingURL** with the name for a map file to the bottom of our JS minified file:

```
//# sourceMappingURL=common.js.map
```

The **sourceMappingURL** property is currently going through a syntax transition due to a problem found with compatibility in IE. So while the most up-to-date syntax is **//#**, some older browser versions may still expect **//@**. In Grunt, we specify our source map in the options for grunt-contrib-uglify as **sourceMap: // path to map file**. In Chrome Developer Tools go to the **Settings** menu and within the **General** tab activate **Enable source maps**. Now when you debug your JavaScript, you can use the un-minified version to see what's really going on, while still providing your minified version to your end users.

20 Try Chrome's Workspace

Chrome has recently introduced Workspace, a highly requested feature that allows you to make changes from the developer tools and automatically update your source files.

To enable Workspace, you'll first need to go to **chrome://flags** and **Enable Developer Tools experiments**. After re-launching Chrome, go to the developer tools **Settings** menu and select the **Experiments** tab. Check the box **File system folders in Sources Panel** before closing and reopening Chrome again.

Back in the **Settings** menu select the **Workspace** tab. Select **Add File System** and add the directory where your files are loaded, giving Chrome permission to access the directory.

Now, go to the **Sources** menu and right-click on one of your CSS or JavaScript files. Select **Map to Network Resource** and from the file list that appears and choose the file that matches your selection. You can now make changes to your CSS and JavaScript from within the developer tools and your files will automatically update.

Keeping updated

Finally, by using the latest browser developer versions, you can use current features and offer feedback on how to make these tools better. Following the links below to ensure you download the latest versions.

- Chrome Canary: http://netm.ag/canary-246
- Firebug Beta: http://netm.ag/bug-246
- Firefox Aurora: http://netm.ag/aurora-246
- IE Platform Preview: http://netm.ag/win-246
- Opera Next: http://netm.ag/opera-246
- Webkit Nightly: http://nightly.webkit.org

Happy debugging! ●

Further reading

There are many more secrets available on the website that accompanies this article, including ways to debug sites on your mobile device. Visit http://devtoolsecrets.com for more information.

The site is hosted on GitHub, so if you know a secret that isn't listed, please submit a pull request.

Official documentation

From Chrome to Safari and IE, each browser provides some documentation for its developer tools:

- **Chrome:** http://netm.ag/ctools-233
- **Firebug:** http://getfirebug.com/wiki
- **Firefox:** http://netm.ag/fftools-233
- **Internet Explorer:** http://netm.ag/f12-246
- **Opera:** http://netm.ag/dragon-246
- **Safari:** http://netm.ag/safari-246

Keyboard shortcuts

A really useful resource to have to hand:

- **List of keyboard shortcuts:** http://netm.ag/shortcuts-246

Older IE support

Unfortunately, older versions of IE do not ship with developer tools, but there are several other options:

- **IE Developer Toolbar:** http://netm.ag/toolbar-246
- **Firebug Lite:** http://netm.ag/firebug-246
- **Companion.JS:** http://netm.ag/comp-246

Extensions

If you want even more functionality from your developer tools, both Chrome and Firebug support extensions. See the following for some popular ones:

Chrome

- **PageSpeed Insights:** http://netm.ag/insights-246
- **CoffeeConsole:** http://netm.ag/coffee-246
- **Tincr (live edit and save to source):** http://netm.ag/tincr-246
- **AngularJS Batarang:** http://netm.ag/angularjsbatarang-246
- **Backbone Developer Tools:** https://github.com/spect88/backbone-devtools
- **Ember Extension:** https://github.com/tildeio/ember-extension
- **Knockoutjs context debugger:** http://netm.ag/contextdebugger-246
- **Grunt Devtools:** http://netm.ag/grunt-246

Firebug

- **FireSass:** https://addons.mozilla.org/en-US/firefox/addon/103988/
- **Pixel Perfect:** https://addons.mozilla.org/en-US/firefox/addon/7943
- **YSlow:** https://addons.mozilla.org/en-US/firefox/addon/5369