

O'REILLY®

Getting Started with the Web



Shelley Powers

Short. Smart. Seriously useful.

Free ebooks and reports from O'Reilly
at oreil.ly/webdev



We've compiled the best insights from
subject matter experts for you in one place,
so you can dive deep into what's
happening in web development.

Getting Started with the Web

Shelley Powers

Beijing • Boston • Farnham • Sebastopol • Tokyo

O'REILLY®

Getting Started with the Web

by Shelley Powers

Copyright © 2015 Shelley Powers. All rights reserved.

Printed in the United States of America.

Published by O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

O'Reilly books may be purchased for educational, business, or sales promotional use. Online editions are also available for most titles (<http://safaribooksonline.com>). For more information, contact our corporate/institutional sales department: 800-998-9938 or corporate@oreilly.com.

Editors: Simon St.Laurent and

Meg Foley

Production Editor: Kristen Brown

Proofreader: Amanda Kersey

Interior Designer: David Futato

Cover Designer: Karen Montgomery

Illustrator: Rebecca Demarest

July 2015:

First Edition

Revision History for the First Edition

2015-06-19: First Release

2015-09-04: Second Release

The O'Reilly logo is a registered trademark of O'Reilly Media, Inc. *Getting Started with the Web* and related trade dress are trademarks of O'Reilly Media, Inc. Cover image © David Merrett "Take off!"

While the publisher and the author have used good faith efforts to ensure that the information and instructions contained in this work are accurate, the publisher and the author disclaim all responsibility for errors or omissions, including without limitation responsibility for damages resulting from the use of or reliance on this work. Use of the information and instructions contained in this work is at your own risk. If any code samples or other technology this work contains or describes is subject to open source licenses or the intellectual property rights of others, it is your responsibility to ensure that your use thereof complies with such licenses and/or rights.

978-1-491-92232-3

[LSI]

Table of Contents

1. Setting Up Your Web Space.....	1
Getting a Domain	1
Using Hosting Companies	7
Communicating with Your Server Using FTP	14
Setting Up an Email Address	21
Quick Review of Web Server Basics	23
2. Structuring The Web Content with HTML.....	27
Basic Page Structure	28
Commonly Occurring HTML Elements	34
3. Styling the Page with CSS.....	43
CSS Stylesheets	43
Applying CSS	44
The CSS	46
Finishing Up	67
4. Going Beyond the Basics.....	73
The New Accessible/Semantic HTML5 Elements	73
Adding Links to Video and Audio Files	80
Using a Static Page Template	84
Adding a Simple Mail-To Form	86
5. Adding Dynamic Content.....	89
Creating a Subdomain For Your Weblog (or Other Purpose)	89
Creating the Wordpress Weblog	90
Exploring Your New Weblog	92

Choosing Your Weblog's New Look	94
Adding the Perfect Site Widgets	97
Adding a Plugin	98
Creating Wordpress Posts and Pages	99
Incorporating Social Media Into Your Site	103
Helping People Find Your Site	106
Adding Weblog Comments	107
6. Advanced Web Technologies and Techniques.....	109
The File Manager	109
Managing Recurring Tasks with Cron	111
Command Line Access with SSH	113
Secure FTP (SFTP)	120
Website Statistics	121
Adding Support for Digital Certificates and SSL	122
Moving Your Site	127

Setting Up Your Web Space

The Web is an increasingly complex place, yet it's never been simpler to create your own place in it. Let's begin by staking a claim for your own web address and posting your very first web page.

Getting a Domain

Your own website establishes a unique online identity untainted by the branding of popular social media tools. No matter the other options for maintaining a space online, including having a Facebook page or a Twitter or Google+ account, nothing represents you, your cause, or your organization better than having your own website. Best of all, it doesn't have to cost a lot of money, and you don't have to be technical or hire professionals to get your site online.

The place to start when creating your website is determining your web address, your *URL*, as it is commonly known. URL stands for uniform resource locator, and is your website's address. The URL is what you type into the address bar in your browser when you want to go to a specific web page, and it's what you use in a link when linking to a story or resource.

The primary component of the URL is the domain name. Google's domain name is "google.com", the Humane Society of the United State's is "humanesociety.org", and the White House uses "whitehouse.gov". All three are similar in that all three start with a descriptive or identifying name—"google", "humanesociety", and "whitehouse"—followed by an abbreviation, ".com", ".org", and ".gov",

respectively. The first part of the domain is the name you pick that best describes your site, followed by a *top-level domain* or TLD, describing the type of website represented by the domain. Combined, both form a unique address that represents your web space.

Before getting into the details about how to find and register your unique domain name, we'll first take a closer look at the TLD, so you can determine which is most appropriate for your site.

The Top-Level Domain

The TLD provides some information about your website, though the semantics behind the more common ones has weakened over the years. For instance, the most frequently used TLD is *.com*, originally intended for commercial uses. However, it has become the de facto, all purpose catchall for domains, used for anything from companies (“oreilly.com”) to food weblogs (“browneyedbaker.com”). Because of such common usage, anyone can use the *.com* TLD.

Most of the TLDs are available to anyone for any use, while others are restricted. The *.gov* TLD is restricted to government use only, as is the *.edu* (education only), and *.mil* (for the military). Other TLDs are open for general-purpose use, but your website must meet certain criteria. These are typically geographically associated domains, such as *.us* for US websites, or *.co.uk* for sites in the UK.

There are many *generic top-level domains*, or gTLDs, available to anyone, including *.info*, *.me*, *.rocks*, and even *.tv*, though the costs for each vary—sometimes considerably. The list of TLDs you can use is long and growing longer by the month:

- *.com*: General-purpose domain, most commonly used
- *.net*: Originally intended for networks, but also used generally
- *.org*: Typically nonprofit organizations, but now used generally
- *.info*: General information
- *.club*: As in “coffee.club”
- *.me*: Assigned to the Republic of Montenegro, but access open to all
- *.photography*: Self-explanatory
- *.rocks*: For the rock star in all of us
- *.guru*: For the self-help experts among us
- *.website*: For those who like redundancy
- *.io*: Indian Ocean, but popular among technology websites

- *.cm*: More open alternative to *.com*
- *.co*: Another open alternative to *.com*

There are now enough openly available TLDs that you should have no difficulty in obtaining an interesting and uniquely you domain.



Sorry, “cats” Is Out

You can choose your favorite name and create a unique domain, unless the name you want is very common, such as “cats”. There are no open domains for “cats”, “dogs”, “money”, and so on.

Registering Your Domain

Once you have an idea of the domain you want, the next steps are to check whether it’s available, and if it is, to register it. Domain name registration ensures that the domain name is yours to use.

You can register a domain name two different ways. The first is to register it through the company you’ll use to host your website. The second is to use a *name registrar*. This is a company that primarily provides name registration services, though many registrars also provide hosting services.

The advantage to registering with your host company is that most provide free domain registration services for a single domain, as long as you remain with the company. And you don’t have to fuss with the mechanics of associating the domain with the actual website.

The advantage to using a name registrar is that it is simpler to transfer a domain if you decide to move your site to a different hosting company. Reputable hosting companies providing free domain services also provide a procedure to move your domain if you cancel your hosting contract. But you’ll usually have to pay a registration fee that’s higher than if you registered the name with a name registrar.

Name registrars also provide more options for maintaining your domain(s), including the ability to *park* the domain until you’re ready to host it somewhere. A parked domain is one that’s held for you at the registrar. It’s a way of reserving your domain until you’re ready to launch your website.

This section assumes you're registering a domain at a name registrar and *parking* it until you find a hosting company.

There are several very good name registrars. Some of the most popular are **Namecheap**, **1&1**, **Name**, and **GoDaddy**. I'll demonstrate the name registration process with Namecheap, though the process is similar in all registrars.

When you access the name registrar web page, the first thing you'll be presented with is a large input-text field where you type the domain you're interested in. The registrar then checks to see if the domain is available. You'll usually type just the name component, not the TLD extension, so that you can see what combinations are available.

Let's say you're interested in a domain name of "blipdebit", as in "blip de bit", not "blip debit". You've picked this name because it's catchy, perhaps maps to the site purpose, and you think it's a unique combination of letters that is available in most, if not all, TLDs. Typing the name in Namecheap's input field returns a result showing us "blipdebit" is available with *all* TLDs, as shown in **Figure 1-1**.

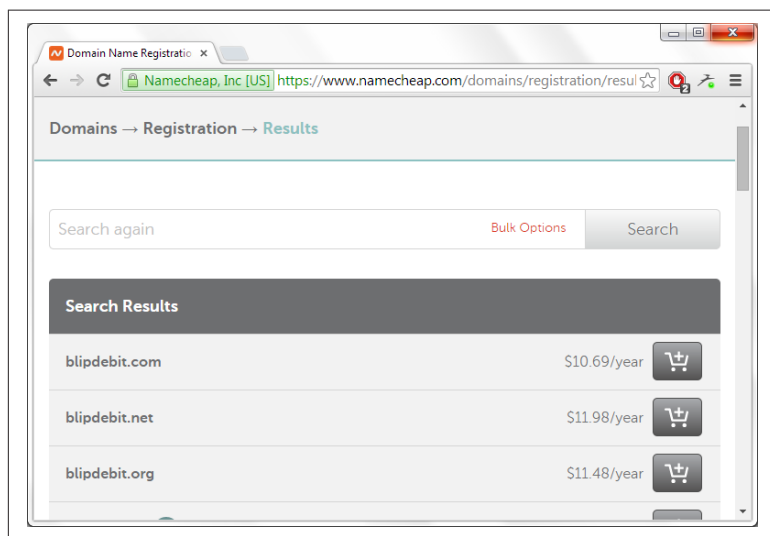


Figure 1-1. Namecheap search results for "blipdebit"

At the bottom of the search results is a brief note about an ICANN fee of \$0.18 per domain (US dollars). All domain names are regis-

tered with ICANN, the Internet Corporation for Assigned Names and Numbers. The small fee covers the cost for this registration.



The ICANN FAQ

ICANN provides a [helpful FAQ](#) and [list of accredited domain registrars](#). Included in the FAQ is more information about the various TLDs, as well as good advice to make sure your registration process is problem-free.

Now you can add as many name/TLD combinations as you wish. I strongly recommend picking *.com*, as this is the most common TLD (the one most people are familiar with). You don't need to select any others, unless you think at some point you'll need a domain name variation for a separate purpose, or you like how the name looks with *.rocks* and want to ensure you have access to it at a later time.

However, if *.com* isn't available, or you just don't care for it, feel free to use whatever TLD is available to you. My own domain, *burning-bird.net*, uses the *.net* TLD, because *.com* wasn't available when I registered it, and I've not had issues with people finding my site.



About Finding Your Site

Nowadays, most people access websites via search engine or links from other sites (or Twitter or Facebook or other social media site) rather than actually typing the domain name into the browser address bar. Later in the book, I'll cover how you can ensure that search engines find your website, and how to promote your site in social media.

After you've added one or more domain names to the shopping cart, you're ready to check out. In the checkout page, Namecheap automatically sets autorenew to off. This means that the domain won't automatically renew with Namecheap when it expires. When you register a domain name, you only register it for a set period of time, typically one or two years. At that point, you'll either need to renew the domain with your existing registrar or move it to a new registrar and renew it. If you don't, you'll lose the domain. You don't have to worry about accidentally losing the domain, as the registrar will provide ample warning of the expiration date.

Some registrars, like GoDaddy, require that you register your domain for two years. Others may set the domain to automatically renew. It's important to read the fine print before checking out so you understand exactly what you're getting.

Namecheap also provides an option to add a WhoisGuard. I strongly recommend you use this, even if you have to pay a small fee (though when I wrote this, the WhoisGuard coverage was free, as shown in [Figure 1-2](#)). ICANN requires that each domain have an associated contact name, phone number, and address. All this information is exposed if you run a WHOIS request for the domain name.

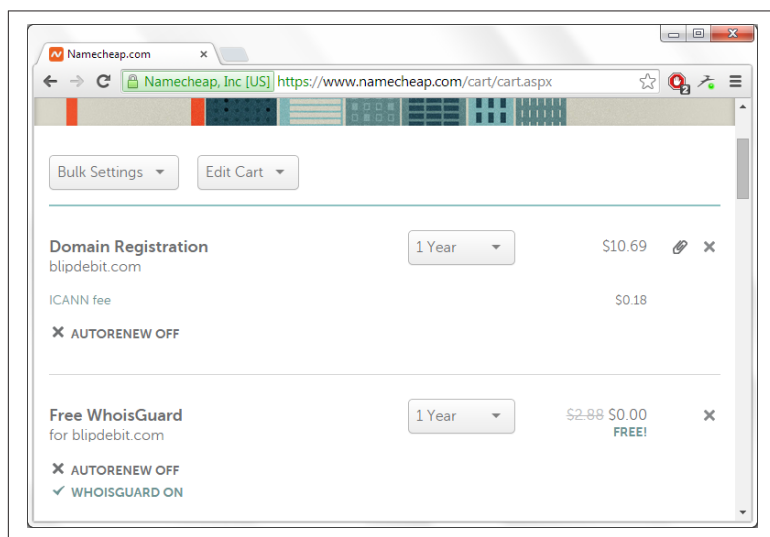


Figure 1-2. Name registration options, including WHOIS guard



WHOIS

WHOIS is a method of searching through the global domain name database for information about an existing domain name. Name registrars use WHOIS to check to see if a domain is available. Typing “WHOIS” in a search engine will return any number of websites that will allow you to check the information about the domain.

As an example, running a WHOIS request on “oreilly.com” at GoDaddy, O’Reilly’s registrar, we discover O’Reilly’s corporate

address, phone number, and website admin email address. Most businesses and large organizations don't care if this information is exposed, but smaller organizations and individuals usually do.

The WHOIS guard service (or whatever is the name of the service provided by the registrar you use) provides a way of sheltering this information. Instead of your name, email, phone number, and address showing up when someone does a WHOIS on your domain, people see ones generated by the WHOIS guard service. People can still contact you...but only through the service. It is more than sufficient to protect your privacy.

The registrar may offer other options, such as support for Secure Sockets Layer (SSL) or hosting your website. I'll cover SSL in **Chapter 6**, but for now, just stick with the domain name and WHOIS guard. Depending on the registrar you use, you might have to wade through several pages of offers before you're actually able to check out.

To finish the registration, you'll need to create an account with the registrar and provide payment information. When you're finished, you're the proud owner of the domain name of your dreams. At this point, the domain name is parked until you sign up with a hosting service.



Domain Name Parking

Namecheap.com does provide a way to customize your parked domain page, but be forewarned that with some registrars, the parked domain page can be filled with ads. If you don't want a page filled with ads for your domain name, and your registrar doesn't provide a way to customize the page, be ready to start at a host, right away.

Using Hosting Companies

A hosting company provides the physical server and Internet access for your website. There are many hosting companies, each providing a set of services. Some companies only provide *shared hosting* services, others are *site builders*, and yet other hosting companies specialize in dedicated servers or *virtual private networks* (VPN).

Types of Web Hosting

A dedicated server is a leased computer maintained at a hosting company website. One of the more well-known dedicated server companies is **Rackspace**. Though the company provides the hardware and Internet access, you're controlling what's installed on the server and are responsible for maintaining it. You can contract with the hosting company to provide setup and maintenance, but it's expensive. The dedicated server itself isn't cheap.

A virtual private server (VPS) is a system where you have all the advantages of a dedicated server, including administrative privileges, but you're sharing a physical server. My own VPS is hosted with **Linode**. You're still responsible for maintaining your slice of the server pie, but it's less expensive than having a dedicated server.

A website-builder host is one where you don't have to have your own domain name. The **Wordpress.com** host is a site like this, as is **Weebly**. Instead of *yourdomain.com*, you'll get a domain like *your-name.weebly.com*. Most site builders provide a free option with very limited functionality, though they also provide an upgrade into a more traditional shared hosting service.

Go For the Cloud

The last few years saw the rise of the cloud, as in *cloud services*. Amazon provides cloud services, as do Google, Microsoft, and many of the hosting companies.

Unlike more traditional hosting services, cloud services allow companies to partition part of their website's functionality to another server. They could host some of their data operations or their more complex processing on the other site.

When looking for a host, you may come across sites offering cloud services. There may come a time when you'll need such a service, but not when you're just starting out.

A shared hosting service is one where your website is one of several running off the same web server. The hosting company manages all of the server functionality for you so you can have both a website and email without having to bother with maintaining the server software. Unless you're proficient with web server software and

operating system maintenance, or can hire someone to do this for you, a shared hosting service is the best bet for you. The next section discusses what to expect when you sign up for a website at a shared hosting service.

Finding Your Hosting Company

You have a wealth of choices when it comes to choosing a hosting company. Following is just a partial list of the companies I'm most familiar with:

- [Dreamhost](#)
- [HostGator](#)
- [BlueHost](#)
- [A Small Orange](#)
- [InMotion](#)
- [Arvixe](#)

To find the one best for you, you can ask friends what they use, explore “best of” online articles related to hosting services, or check out who shows up at the top of the search engine results. In each case, you'll want to check out reviews for the company and ask folks you know if they know the company and if they like it. Just be forewarned that people either love their hosting companies unconditionally, or loathe them, so any personal reviews you get can be skewed.



Reviews of Best Hosting Companies

PC Magazine put together a great [side-by-side review](#) of several different hosting companies, providing a good starting point in your company search.

Shared hosting systems are typically Linux-based, though you can find hosts that support Windows. I recommend a Linux website even if you're most familiar with Windows, as Linux is the most common type of system with support for the largest number of software and service options. It's also the less expensive option, and you don't have to touch the Linux operating system until you're ready for more advanced functionality—all of the basic website management functionality can be managed via a control panel. I would suggest

only going with Windows if you really want to use Windows-specific functionality, such as ASP.NET.

Each hosting company offers packages that contain support for various website services, including bandwidth (most offer an unlimited amount), storage space, domain and subdomain support, pre-packaged shopping carts, mailboxes, spam filters, and control panel support.



Subdomains

A *subdomain* is a subset of a domain. If you've seen URLs like `http://doc.somecompany.com`, or `http://technology.anothercompany.rocks`, the “doc” and “technology” represent subdomains. These subdomains can use different applications and even have more restricted access. They're a way of partitioning your site while still only having one domain.

What package you choose depends on your needs. If you want support for more than one domain, you'll need to ensure the package allows for multiple domains. If you want email support with your domain, the package needs to provide mailbox services. If you're putting together a small, online store, having access to shopping cart functionality is essential. FTP support is a must, as is database support if you plan on eventually using weblogging software or a content management system (CMS).

Regardless of your unique needs, one service you must have is a control panel. This is an online interface that allows you to easily control your site, as well as set up the individual services. I strongly recommend that you get a shared host package with support for **cPanel**, the most frequently used (and well documented and supported) control panel.



Plesk Control Panel

Another well-known control panel is Plesk. Though I cover cPanel in this book, the procedures when using Plesk should be similar.

In the next section, we'll take a look at setting a site up using Blue-Host as the shared hosting company.

Signing Up with a Host

Focusing in on one host, BlueHost, I'll walk you through the sign-up process and how to connect your new domain name with your new website. As with the name registrar, signing up for a hosting service is similar across all of the companies.

Most hosting companies provide different packages, and BlueHost is no exception. It provides three options for shared hosting: Starter, Plus, and Pro. Note that the price quoted is if you sign up for a three-year package, so be forewarned that the price per month is higher if you decide on a 12- or 14-month package.

Since you're just starting, and you only have one domain, let's go with the Starter package. You can always upgrade at a later time (most hosts allow upgrading, so start cheap, and work up). In the sign-up page, you'll either provide a domain for BlueHost to register for you or type in the domain name you registered with a name registrar. Since you've already used a name registrar to park a domain, go with the second option.

The next page collects account information, including name and address, as well as payment information. With rare exceptions, shared hosting companies require payment upfront for at least a year's worth of hosting.

BlueHost, like the other hosting companies, also provides optional extras you can add to your account, including enhanced backup support and site security. For now, unless there's an option you're absolutely sure you want, go with the basic service (you can add most extras at a later time).

The last page presents a receipt and asks you to provide a password for accessing the new website. Be sure to record this password. You'll then be taken to the login page, where you can log in using your domain name.

Congratulations, you're all signed up. Now comes the fun part.

Your Website's Address and Connecting Your Domain

You have a domain managed by one company, and your website hosted in another. How do you connect the two? Easily. All you need to do is point your domain to BlueHost's *name servers*. Before

we make the connection, though, let's take a closer look at how domains get mapped to server locations, and what a name server is.

The Internet addressing system consists of *Internet Protocol* (IP) addresses that usually look like four sets of three numbers, separated by periods. My site's current IP address is *173.255.206.103*. You can type this address into a browser and my default website displays, since I have a *dedicated IP address*. The system for the address is IPv4, which has been the primary Internet addressing system for years. Recently a new addressing system, IPV6, was created because the popularity of the Web is depleting IPv4 addresses. An example of an address under IPv6 is:

```
FE80:0000:0000:0000:0202:B3FF:FE1E:8329
```

Definitely not for the faint of heart. And not something you want to type into a browser address field. Or try to remember. Neither address system provides addresses that are easy for humans to remember.

What we need is a way to map these network addresses to domain names humans do understand. That's where the *Domain Name System* (DNS) enters the picture. The DNS maps network addresses to domain names via a hierarchy of name servers that maintain this pairing.

BlueHost provides two name servers: *ns1.BlueHost.com* and *ns2.BlueHost.com*. In these name servers, the company provides the information that maps the domain name you provided when you signed up with the IP address you share with other BlueHost customers. All you need to do now is transfer the DNS management for your parked domain at the name registrar to the hosting company. You do that by replacing the name servers the registrar lists with the name servers the hosting company provides.

Returning to Namecheap, look for the menu option *Manage Domains*, listed in the menu under your account name. In the left side of the page is a *View Domains* option. Clicking this will list all your domains to the right. Click the domain you're having BlueHost manage. Select "Transfer DNS to Webhost" from the left side in the page that opens, as highlighted in **Figure 1-3**. In the page that opens, you'll see five text fields in the middle of the page. Enter the web host's name servers (in this case, *ns1.BlueHost.com* and *ns2.BlueHost.com*) into the fields, as demonstrated in **Figure 1-3**. You need to

enter at least two, but you can enter up to five if your hosting company provides more than two name servers.

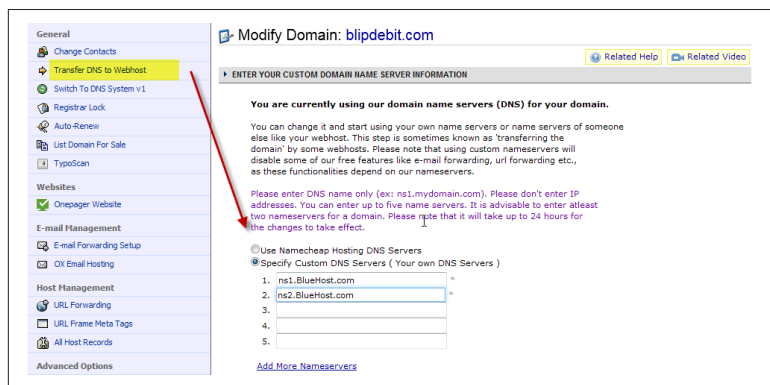


Figure 1-3. Selecting Transfer DNS to Webhost opens a page where you can enter the host's name servers

That's it. You're all done. The same process works regardless of web hosting company and name registrar:

1. Register your domain at the name registrar.
2. Sign up with a web host, providing the domain name when prompted.
3. Find the web host's name server names. Typically these will be available via site documentation or FAQ. Or you can search for the name of your host and "name server."
4. Transfer DNS from the registrar to the host by copying the name server addresses into the spaces provided at the registrar.

Again, if your host is also your registrar, you can skip the DNS transfer process. But if you want to switch to another host, you'll have to go through a DNS transfer at that time.

If you enter the domain into your browser once the DNS is transferred, your website may not show up right away. It can take a day or two for the transfer to *propagate* throughout the name server hierarchy. An interesting effect of the propagation process is that a friend or family member might be able to access the domain at your website before you can.



Follow Your Domain Name

Domain propagation is the process where the domain/IP address makes it way throughout the Internet. You can follow your domain name as it propagates through the DNS in the [Global DNS Propagation Checker](#). Checking your domain over time, you'll see the IP address associated with your domain slowly change from the registrar's IP address to your web host.

When the DNS transfer is complete, the host's generic hosting page shows up. This page displays until you upload your own content, covered next.

Communicating with Your Server Using FTP

You'll need to use *file transfer protocol* (FTP) to upload content to your new website. You can create new FTP accounts via your cPanel control panel.

Introducing the cPanel

Most hosting companies use cPanel, the most popular of control panels. However, not all cPanel installations look the same, because the software allows each hosting company to brand the tool with its own look and feel, as well as refine what options are displayed for its customers.

For the most part, though, if you've used cPanel at one company, you can easily use it at any other. Or if you've used the cPanel company's own [demo installation](#), you can easily use the hosting company's version.

Logging back into BlueHost (or whatever hosting company you're using) using the [direct home link](#), your account's cPanel management system is displayed, as shown in [Figure 1-4](#).

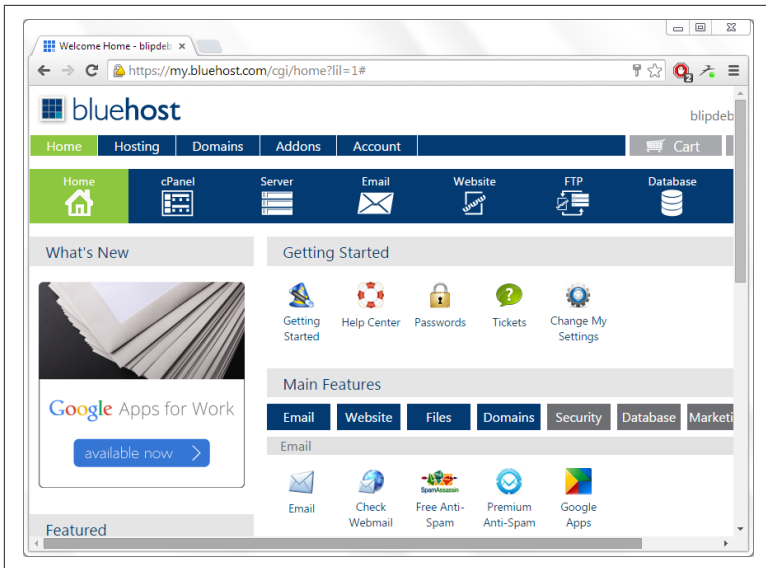


Figure 1-4. The cPanel home page at BlueHost

Across the top are several options, including access to the Server. Other cPanel installations may not have the topmost set of links, in which case you'll need to look for the service icons within the cPanel page. Each should be clearly labeled.

When you click the Server option, BlueHost asks you to verify your account for the first time. Hosting companies have been trying to shut down spammers with extra steps, and this is one of them. It's a pain, but thankfully, you only have to verify your account once (this process does differ across hosting companies). Verification is easily accomplished by calling a given phone number using the phone associated with your account, and providing the last four characters of your password or last four digits of your credit card.

Once verified, clicking the Server option opens a page with links to check your disk space usage, bandwidth usage, and so on. It's a way to keep track of what's happening with your website. The other cPanel icons along the top (e.g., Email, Website, FTP, Databases, and Manage IPs) are links to options located elsewhere in the cPanel page. They're ways of managing website software, email accounts, FTP logins, databases, and unique IP addresses, if you ever purchase an individual IP address.

Clicking the FTP option opens a page that prompts you to create an FTP account, which I'll cover in the next section. I'll cover other cPanel options in later sections and chapters.

Setting Up and Using an FTP Account

When you first set up your website, unless you're using packaged software such as Wordpress, you'll need to upload at least one web page. You'll need an FTP account in order to use FTP to transfer the page from your home computer to the server.

An FTP account is a way of using client-side software to create a connection between your computer (or tablet or smartphone) to your website on the hosting company's server. You can use any FTP software, as long as it supports the type of FTP you're using. By default, an FTP account is nonencrypted, which means that any data you send or receive from the server is sent without encryption and can be seen by any hacker who may have tapped into the system. In **Chapter 6** I'll cover how to set up SFTP, or *secure FTP*, for your account. SFTP adds encryption so that any snoop can't peek at the data. For now, though, we'll stick with FTP, just to get you started.

When you signed up for BlueHost, the company created a default FTP account for you. You'll see this FTP username in the welcome email. The password is the same password you created to access cPanel. The email also provides the FTP URL for your site, which is the domain name with an *ftp* subdomain. For *blipdebit.com*, it's *ftp.blipdebit.com*.

Any FTP client software can provide access to your server. The one I use is **Filezilla**, a free application that works in all platforms. Once downloaded and installed, opening the application provides a page listing a local subdirectory (on your computer) and files in that subdirectory in windows to the left, and directories and files on your server to the right. To establish a connection to your server, you'll either type in the host name, username, and password at the top (i.e., *ftp.blipdebit.com*, *blipdedi*, and whatever password), or you can click Site Manager in the File menu. Using Site Manager allows us to create persistent accounts for the servers, rather than having to re-type the account information each time we want to connect. **Figure 1-5** shows the Site Manager entry for *ftp.blipdebit.com*.

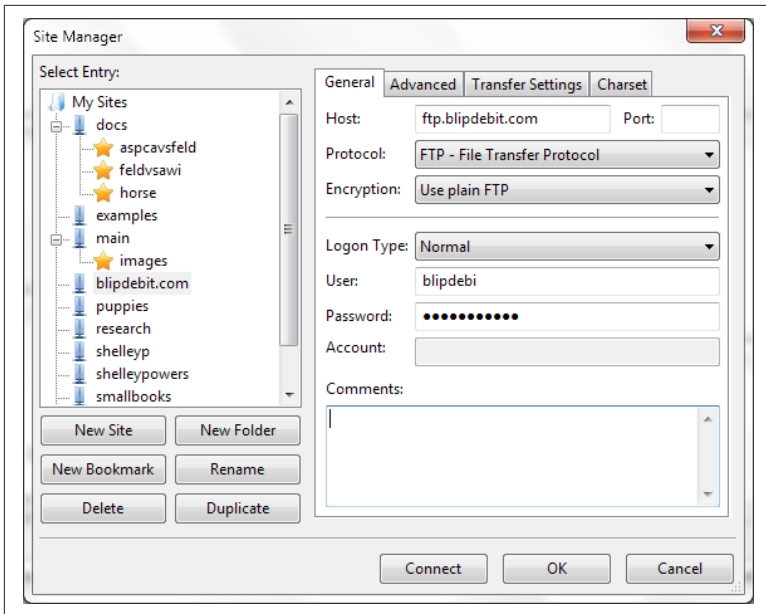


Figure 1-5. FileZilla Site Manager entry for *ftp.blipdebit.com*

You don't have to specify the port when you create the entry, because FileZilla will use the default port (21) for FTP. I'll cover the concept of ports later in [“Quick Review of Web Server Basics” on page 23](#).

Ports: The Hidden Address Component

What isn't shown in the URL is the *port number*. A port is a communication endpoint: the connecting component of any network communications. Your server can serve up pages for several different ports, as well as several different domains and subdomains. The default port for web pages is port 80. Because it's the default, browsers and applications that can process requests for web pages without you having to specify the port—they assume port 80 by default. You can, though, actually provide the port if you wish:

```
http://somedomain.com:80/some-article
```

Note that the port is separated from the domain by a colon.

Different services have different default ports. HTTP Secure (HTTPS), covered in [Chapter 6](#), has a default port of 443. FTP uses 21, while SFTP uses 22. Email's default port is 25.

A second way you can configure your FTP client is download the FTP client configuration files that cPanel provides for each FTP account. In the FTP Accounts page, click the Configure FTP Client links associated with an account. A page section opens beneath the account, with links to three different FTP clients: Filezilla, Core FTP, and Cyberduck. Click the one you wish, and save the file to your local PC. To use, open your FTP client and import the configuration file. For Filezilla, do this by selecting File, and then Import. A new site configuration is created automatically for you.

Clicking on Connect creates the connection to your server. If you used a configuration file, you'll be prompted for your password. The right side of the FileZilla window now shows the subdirectories and files in your server. The default FTP account created when you created your hosting account opens the server connection at the top-most level of the file hierarchy for your account. Just like in Windows, the Mac OS, or any other environment, file subdirectories on the server are created in a tree-like hierarchy. The subdirectory you'll focus most of your effort in is named *public_html*, as shown in [Figure 1-6](#). The files in this subdirectory are publicly accessible by people accessing your website.

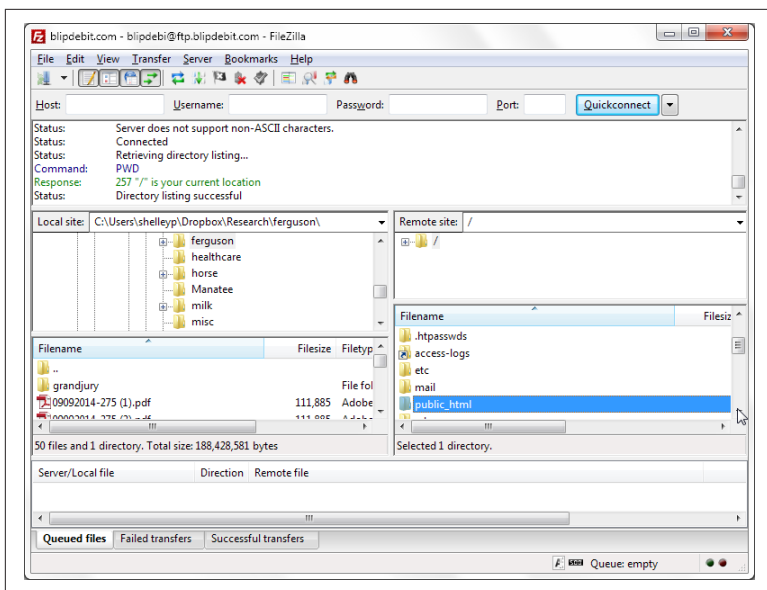
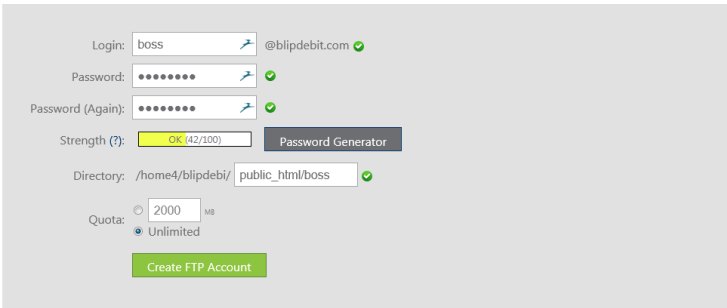


Figure 1-6. The blipdebit.com website opened in FileZilla

You can transfer files from your computer to the server, or transfer files from the server to your computer, once you connect to the server. You can transfer web pages, graphic files, or other document types that you want to make accessible from you server.

If you want the web page or other document to be publicly accessible, you'll need to place it in the *public_html* subdirectory. This subdirectory forms the root for your website. In fact, since most of your work is in *public_html*, it's handy to have an FTP account that opens directly into *public_html*, rather than at the higher account level. It's a simple matter to create a new FTP account.

In the cPanel page that opens when you click on the FTP icon, you'll see a form for entering information about the FTP account. Type in whatever name you'd like to use for the FTP account and a password. Note that the account name is for a specific use, not necessarily a person. When you create an FTP account, the system automatically provides a separate subdirectory for that account name, so you can upload files directly to the subdirectory. If we type in "boss" as the account name, the form automatically creates a new subdirectory reflecting the account name, as shown in [Figure 1-7](#).



The screenshot shows the 'Create FTP Account' form in cPanel. The 'Login' field contains 'boss' and has a dropdown arrow and a green checkmark next to '@blipdebit.com'. The 'Password' and 'Password (Again)' fields are masked with dots and each has a dropdown arrow and a green checkmark. The 'Strength (?)' field shows 'OK (42/100)' in a yellow box, with a 'Password Generator' button to its right. The 'Directory' field shows '/home4/blipdebi/' followed by a dropdown menu showing 'public_html/boss' with a green checkmark. The 'Quota' field has two radio buttons: '2000 MB' (unselected) and 'Unlimited' (selected). A green 'Create FTP Account' button is at the bottom.

Figure 1-7. Creating a new FTP account

Since we want the account to load files directly to *public_html*, we'll delete the added *boss* subdirectory. Clicking Create FTP Account creates the new FTP account. When you use this FTP account in your FTP application, you'll need to use the full FTP account name of *boss@blipdebit.com* for the username. Once connected, the software opens directly into the *public_html* directory. Now you're ready to upload your first web page.

Uploading Your First Web Page

The BlueHost generic page is better than an error of 404 Not Found, which is what people get in their browsers when they access a web page that doesn't exist. But you're really going to want to start putting up your own content.

We're going to start small. Using your favorite text editor, copy the following text (including angle brackets and other annotation) into a new document, and name it *index.html*:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>The HTML5 Herald</title>
  <meta name="description" content="Blip de Bit!">
</head>

<body>
<h1>Hello, World!</h1>
<p>How do you like my new site, eh?</p>
</body>
</html>
```

The text is Hypertext Markup Language (HTML), which is the language used for most web pages. This specific web page is based on the latest version of HTML, HTML 5. I'll cover HTML in more detail in **Chapter 2**, but this page basically creates a plain web page with “Hello, World!” in large text, and “How do you like my new site, eh?” in smaller text underneath.

Open your FTP client, connect to the server using your *public_html* account, and transfer your new HTML document to your server. Now when you open your new website, you'll see a web page like that shown in **Figure 1-8**.

Congratulations! You're now an official webmaster. And since people want to send email to webmasters, let's set up an email address to use.



Figure 1-8. Your very first web page

Setting Up an Email Address

Nowadays, people are more likely to use their Gmail account for all site-related email needs, but it's simple to set up an email address if you want one associated with your domain. In fact, it's not unusual for websites to have a *webmaster@domain.com* email for people to send emails about potential problems at the site (or to spam you, unfortunately).

Click the Email icon in cPanel to create a new email account. In the page that opens, enter the information for the email account, including email username, password, and the size of the email mailbox, as shown in [Figure 1-9](#).

A screenshot of a web form for creating an email account. The form has a light gray background. It includes the following fields and controls: 'Email:' with 'webmaster' entered and '@ blipdebit.com' with a green checkmark; 'Password:' with masked characters and a green checkmark; 'Password (again):' with masked characters and a green checkmark; 'Strength (why?):' with a green bar indicating 'Strong (80/100)' and a 'Password Generator' button; 'Mailbox Quota:' with a radio button selected for '250 MB' and an option for 'Unlimited'; and a green 'Create Account' button at the bottom.

Figure 1-9. Creating your webmaster email account

Once you've created the new email address, it's listed in the Email accounts table at the bottom of the Email page. You can change the email account's password or mailbox quota by accessing the email account in this table.

You can also access the account's email. At the end of the table is a drop-down menu from which you can choose to either configure an email client, or access Webmail: an online web email client built into cPanel that allows you to manage your email directly in the web page.

In the page that opens, you'll be given options to access your email using various web-based email clients, as shown in **Figure 1-10**. You can use any of the clients you'd like, or you can set up your email to automatically forward email to your Gmail or other account. You can also configure client email software from this page. For now, let's access the Squirrel web-based email client. I've always been fond of small, furry rodents.

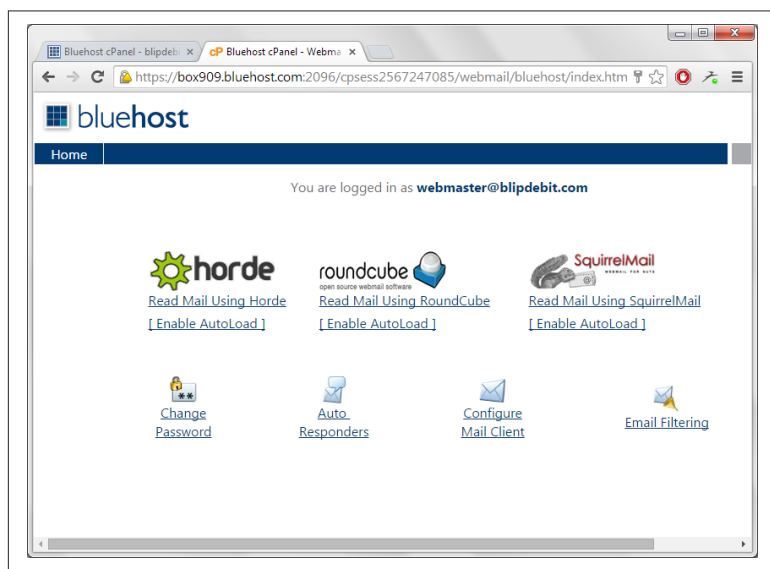


Figure 1-10. The various web-based email clients available to you from cPanel and other options

You don't have any email, of course. From whatever email system you use now, go ahead and send yourself an email. You'll see the new email show up in your email client almost immediately. Clicking on the email opens it in the client, as shown in **Figure 1-11**.

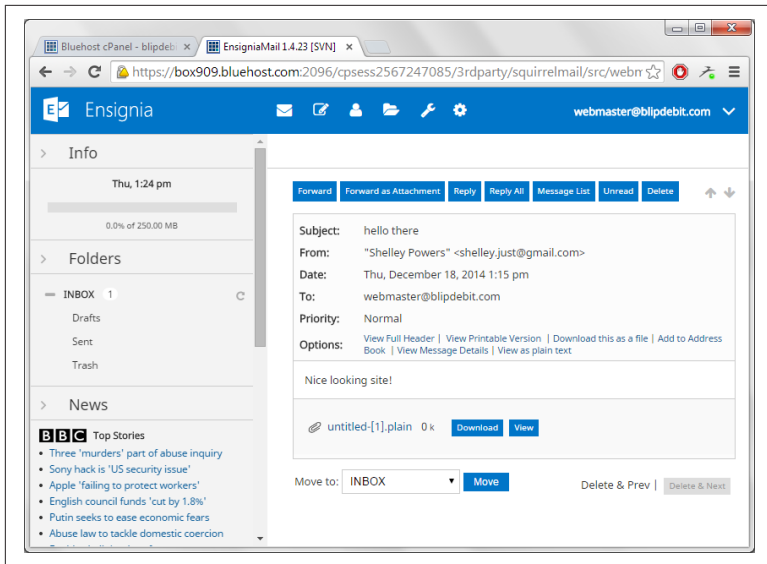


Figure 1-11. Your first webmaster email

Quick Review of Web Server Basics

We covered a lot of ground in one chapter.

You started out by registering a domain with a name registrar. You then signed up with a new web hosting company and transferred DNS management from your name registrar to your hosting company.

You also created your first FTP and email accounts, as well as uploaded your very first web page and sent an email to your new email account.

I wanted to briefly touch on some of the technology behind the scenes of all this activity; just enough so that if people mention any of it, you'll know what they're talking about.

Your domain name is a combination of name and TLD, or top-level domain. It's part of the URL you type into a browser address bar. The other components of the domain are the protocol used, typically HTTP (hypertext transfer protocol), and possibly additional text following the domain for a specific website article. An example is:

`http://www.somedomain.com/some-article`

The *www* in the domain name used to indicate URLs in the World Wide Web, but such use isn't necessary nowadays. It's more of an artifact than a requirement, and many sites completely forgo its use. The domain name, and the *www* subdomain should point to the same site if the hosting company configures the domain setting correctly.

We also briefly touched on the FTP protocol, used for transferring files to and from the server. Though we didn't mention it, email is dependent on yet another protocol: the simple mail transfer protocol, or SMTP.

Moving on from the URL and protocols, let's take a look at your physical website server. Your hosting company is likely using the most commonly used web server there is: Apache. It's ideal for shared hosting because it allows hosting companies to create many different domains off the same server. In addition, the hosting company we covered in this chapter is using Linux-based servers. Again, this is the most common type of servers for web companies. One major advantage to the Linux system is it allows the host company to host several customers on the same server, yet keep each of your files safe and private from the other customers. The host company also has a plethora of tools to monitor what's happening in your site, many of which you can access directly in cPanel via the System link I covered earlier.

Your web pages are served up by Apache, but different software is used for FTP as well as handling email. Email servers are some of the most complex servers to set up, and a major reason many people really don't want to maintain their own web-based systems. Which email server your host is using is, thankfully, something you'll never have to deal with.

In the background are a host of other services busy keeping your site safe and secure. There are firewalls to keep the bad guys out, and backup programs to help you recover your site if it gets trashed. Most hosting companies provide good, basic security, spam protection for email, and backups for free, but if you want to have more in-depth control, you'll most likely need to purchase an optional service. Again, these services should be available on cPanel, and you'll recognize if they're for pay or not by the addition of "Pro" to the icon title.

If you run into problems, you can check out the hosting company documentation. Most provide excellent documentation, how-tos, and FAQs. In addition, you can open a ticket via cPanel and either submit a question, note a problem, or ask for help. Be aware, though, that hosting companies won't manage your website for you. But then, that's why you have a book like this: to understand how to manage your first website yourself.

Structuring The Web Content with HTML

Now that you have your hosting company and your very first web page, it's time to take a closer look at exactly what it is you uploaded.

The page you uploaded in [Chapter 1](#) consisted of the markup shown in [Example 2-1](#).

Example 2-1. Your first web page

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <title>Blip de Bit</title>
    <meta name="description" content="Blip de Bit!">
  </head>
  <body>
    <h1>Hello, World!</h1>
    <p>How do you like my new site, eh?</p>
  </body>
</html>
```

In this chapter, I'm going to cover the very basic HTML elements in the page, as well as some others you'll see used universally throughout the Web.

Basic Page Structure

Hypertext Markup Language, or HTML, is the language of the Web. It consists of *elements*—text components enclosed in angle brackets (<, >), each of which has its own meaning and default appearance. Browsers have the ability to read the markup elements, such as <head> or <p> and know exactly what to do with the contents. Some of the elements provide information; others provide structuring for the page contents.

In [Example 2-1](#), the basic page structure is defined by four overall elements:

- The **doctype** declaration
- The **html** element
- The **head** element
- The **body** element

The doctype

The **doctype** element (<!DOCTYPE html>) tells the browser what type of markup is being used in the page. In [Example 2-1](#), the markup is HTML5, the fifth version of the most common type of HTML. Other common **doctype** declarations are ones for HTML 4.01 transitional:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
```

and XHTML 1.1:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML Basic 1.1//EN"
"http://www.w3.org/TR/xhtml-basic/xhtml-basic11.dtd">
```

The **doctype** is important because it gives the browser a heads-up about what to expect, and also how to treat the individual markup elements. Take the XHTML 1.1 **doctype**. XHTML, or Extensible Markup Language, is another variation of web page markup. The rules governing its use are much stricter and less forgiving of mistakes (which is one reason it's not as popular as HTML). If you don't adhere to the XHTML rules, the page just won't display. On the other hand, HTML5, which we're using in the book, is very forgiving of minor mistakes, as long as you use the correct spelling for the element tags.



Lowercase Versus Uppercase Elements

In [Example 2-1](#), **doctype** is in all-caps, but the other HTML elements are lowercase. HTML5 works with both. However, the common usage is all lowercase elements. My use of all-caps for **doctype** is more of a habit than a recommendation.

The html Element

The first element after the **doctype** in [Example 2-1](#) is the **html** element. It's also known as the document's *root element*. It's not a required element, but, as we'll soon see, it is very useful.

In the example, the **html** element consists of *opening* and *closing* tags:

```
<html lang="en">...</html>
```

The opening tag consists of the element's name (and any attributes—more on these later), enclosed in angle brackets. The closing tag is also enclosed in angle brackets, but the element name is preceded by a forward slash (/). The closing tag isn't mandatory in HTML5, as it is for other document types (e.g., XHTML). In HTML5, tools that process the HTML documents use cues, such as open tags for other elements, as indicators it should treat the element as closed. The closing cue for the **html** element's closing is the end of the web page. If you do use the closing tag, it must be the last piece of content in the web page.



Use Closing Tags

It's a good idea to use closing tags, if for no other reason than ensuring no errors that can impact the page's visual display are introduced. It also helps to keep your markup visually organized.

The **html** element has content within the opening tag. This content is known as an *attribute*. HTML attributes provide additional information about the element, separate from its content. In [Example 2-1](#), the **html** attribute is `lang="en"`. It lets the browser know that English is the primary language used in the web document. Browsers are usually smart enough to figure this out, but it helps to be as precise as possible in the markup. The more informa-

tion you provide, the less the browser has to guess (and possibly guess incorrectly).

HTML is a hierarchical language, which means that elements are contained within elements, and the overall document forms a tree-like structure of nested elements. The **html** element wraps all of the other web page elements, some of which also contain nested elements, and so on. The first **html** nested element is the **head** element.

The head Element and an Introduction to Empty Elements

The **head** element isn't required, but I strongly recommend you include it, if for no other reason than to give your web page a title. The **head** element is used to organize the *metadata* for the web page. Metadata literally means data about data, and the metadata in the **head** element provides information about the web page.

The most common of the elements nested in the **head** element is the **title**. The **title** element provides a title for the web page. It may be the same as the web page's article title, but it usually reflects the website name in some form. In [Example 2-1](#), the title is Blip de Bit. If you open the website in a browser, the title displays in the tab at the top of the page. [Figure 2-1](#) shows Firefox with two opened tabs: one to [blipdebit.com](#), and one to another of my sites, [burningbird.net](#).

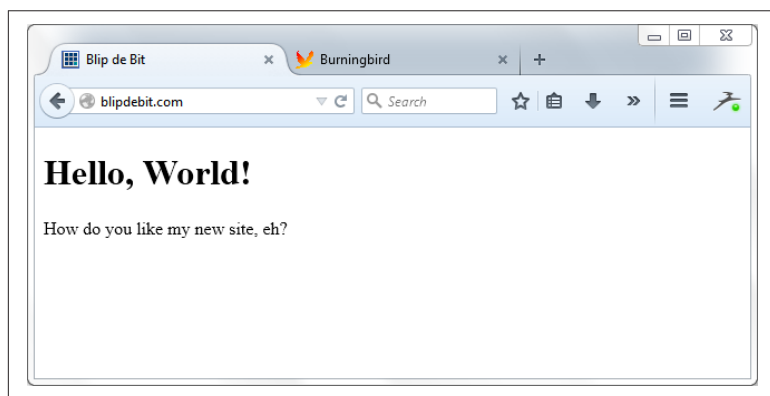


Figure 2-1. Web page title elements on display

The second metadata element is **meta**. It's a generic element for marking metadata that isn't marked with a custom element. In the

example, the **meta** element is used to provide two pieces of information:

- **Charset**—the character encoding used for the document
- A web page description, given with the **name** and **content** attributes

The character encoding for a web page document tells the browser what kind of encoding scheme is used for the special characters, like the copyright ©. When the browser encounters a character sequence, such as ` ` or `©`, it knows to display a single space or the copyright character, respectively. The most common character encoding for the Web is *UTF-8*, as shown in [Example 2-1](#).

The **name** and **content** attributes in the second meta element are used to designate name/value pairs permitted in HTML5. From the W3C HTML5 specification, the following are allowed:

application-name

When the **name** attribute is set to `application-name`, the value of the **content** attribute must be a string representing the name of the web application that the page represents.

author

When the **name** attribute is set to `author`, the value of the **content** attribute must be a string that gives the name of one of the authors of the document.

description

When the **name** attribute is set to `description`, the value of the **content** attribute must be a string that describes the page.

generator

When the **name** attribute is set to `generator`, the value of the **content** attribute must be a string that identifies the software used to generate the document.

keywords

When the **name** attribute is set to `keywords`, the value of the **content** attribute must be a set of comma-separated strings, each of which is a keyword relevant to the document.

The example is providing a page description using the **meta** element.



Myths About Meta

A common myth about the **meta** element is that it can aid your search engine results. Search engines have progressed beyond the need of utilizing the **meta** element for useful information.

Notice that the meta elements don't have any closing tags. They wouldn't regardless of **doctype** used, because they're known as *void*, or *empty*, elements. In other words, they are elements whose function comes in via their attributes, not their contents.

Another way empty elements can be displayed is as self-closing tags:

```
<meta charset="utf-8" />
<meta name="description" content="Blip de Bit!" />
```

The element tags have been modified with an ending backslash (/). The self-closing tag markup is required with XHTML, but not HTML5. You can use the self-closing markup in HTML5, but it isn't required and is rarely used nowadays.

That's the last of the elements included in the **head** element for [Example 2-1](#). Later, we'll see some other commonly occurring elements. For now, let's finish off the rest of the page structure.

The body Element

The **body** element contains the web page contents. In [Example 2-1](#), the contents consist of an **h1** header element and a paragraph (**p**) element. The contents in the two elements are displayed in the page, but as was demonstrated in [Figure 1-8](#) in [Chapter 1](#), how the contents are displayed differs.

The HTML specification provides minimal display characteristics for the HTML elements. The different header elements, ranging from **h1** all the way to **h6**, are bold by default, have a specific font type and size (decreasing in size from **h1** to **h6**). More importantly, they have a semantic meaning. When search engines parse the page, they can recognize the text that forms the heading for a story or article. When screen readers interpret the page, they can determine which text should be spoken, first. You could use a paragraph as a header, and use CSS (discussed later) to style it to look the same as a header element, but then you'll lose the semantic advantage of using the proper element.

The same applies to the paragraph (**p**) element. Like the header, it has a default styling. It inherits the default font type and size defined in the default browser *stylesheet*—the document that contains the styling instructions for the page contents. It's also a *block element*, like the headers, which means that the elements expand to fit the width of the space containing it and begin a new line at the point where they're inserted. In the following code block, two headers are divided by two paragraphs:

```
<h1>The Big Header</h1>
<p>The h1 header is equivalent to a chapter title, while the
headers (h2 through h6) progressively nested subtitles and
section headers.
</p><h2>The Sub-Heading</h2>
<p>Notice how each if you have two elements, one following
the other, how you end one element, first...</p>
<p>...before starting another</p>
```

Figure 2-2 shows the page contents, as loaded in Chrome and using the default Chrome stylesheet. Even though the **h2** header starts on the same line as the paragraph preceding it, it begins a new line in the actual display. The same with the paragraph elements.

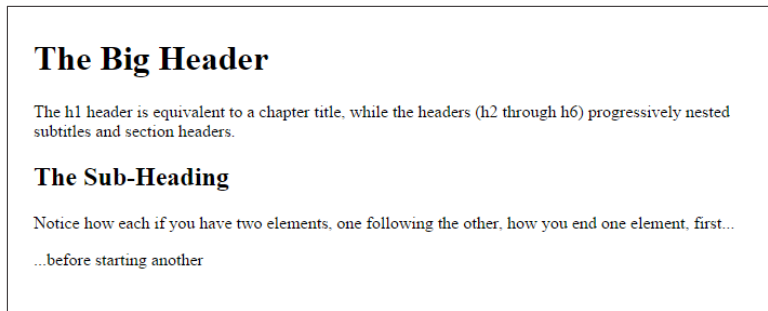


Figure 2-2. Two headers (*h1* and *h2*) with three different paragraphs

Some elements can nest in other elements, as the **>body** and **>head** elements do in **>html**, and the **>h1** and **>p** elements nest in the **>body**. But when the elements are nested in another, the entire element must be contained in the outer element, and that includes both the opening and closing tags if you use both. >

```
<body>
  <p>paragraph...</p>
</body>
```

If you use closing tags, the closing tag for the paragraph must occur before the closing tag for the **body** element.

You can do a lot with just headers and paragraphs in a web page, but you're going to need more at some point. Some of the more commonly used HTML elements are introduced in the next section.

Commonly Occurring HTML Elements

There are a large number of HTML elements in addition to those we've covered, but there are a core set you'll frequently encounter. We'll take a look at these commonly occurring elements in this section. HTML5 introduced new categorization schemes, but most of us split the HTML elements into as *block-level* elements, or *inline* elements, which I'll cover first.



Newer Semantic Elements

The HTML elements I cover in this chapter are all old friends who have been with us past editions of HTML. I cover newer, semantically rich elements introduced in HTML5 in [Chapter 4](#).

Inline Elements

As you might expect with the name, inline elements are those are used within blocks of text, to modify or add additional information to sections of text within the block.

The most common inline element has to be the link, the **a** element. Without it, the Web wouldn't be the Web. Links are what we use to connect web pages to other web pages. A web page link is a very simple thing. The information about the link is defined in attributes, while the text the link surrounds is displayed as the link text:

```
My publisher is <a href="http://oreilly.com">O'Reilly</a>.
```

By default, the text enclosed in the link element is underlined, and offset with a different color (typically blue by default). Clicking the link opens the web page defined in the **href** attribute.

Another very popular inline element is the **img** element, which allows us to embed an image in the web page. All browsers support PNG, GIF, or JPEG image files, while some support other types,

such as SVG (scalar vector graphics). An example of an **img** element for a JPEG is the following:

```

```

The URL for the image is given in the **src** attribute. In the HTML, the image source file has already been uploaded to your site, and the location of the image is relative to your site's primary content folder. If your main website pages are loaded into the subdirectory named *public_html*, and you create a new subdirectory named *images* in *public_html*, then the image URL would be *http://yourdomain.com/images/image.jpg*.



Avoid Hot Linking

You can link images in other websites, a technique known as *hot linking*, but doing so is frowned on, and many sites block others from directly linking their images.

The **alt** attribute is a brief description of what's displayed in the image. It's used to provide a textual description of the contents for people using screen readers or other assistive technologies. It's not mandatory to provide the **alt** attribute, but it is strongly encouraged.

Also note that the **img** element is an empty element. All of the necessary information comes from the attributes. I used the end forward slash in the element, but you don't have to include it as long as your document is HTML5.

Several inline elements provide textual semantics, as well as some default formatting. Among the most popular are the **strong**, **b**, **em**, **i**, and **small** elements. They're wrapped around the text you want to highlight in some way.

This is a paragraph and I want to `emphasize this text`.

You do need the ending tag for these inline elements, as well as the other inline elements that are not empty.

The **strong** element is used to indicate serious or important material. By default, it's portrayed in bold text. If you're only interested in bold text, and aren't interested in highlighting text because it's essential or important, you can use the **b** element instead.

The **em** element is used to emphasize the text. It's displayed as italic text, by default. If you just want to display text in italics, and aren't interested in attaching semantic importance, use the **i** element, instead.

The **small** element is used to designate small print, such as copyrights, contact information, legalese, and so on. By default, it's displayed in a smaller font than the other web page text.

Example 2-2 utilizes all of the inline elements we've covered by modifying our original HTML document and adding new content.

Example 2-2. Blip de Bit web page modified with new inline elements

```
<!doctype html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>Blip de Bit</title>
  <meta name="description" content="Blip de Bit!">
</head>

<body>
<h1>Hello, World!</h1>
<p>How do you like my new site, eh? </p>
<p>I want you to feel <em>welcome</em> when you visit my website.
I don't <strong>ever</strong> want you to feel you came to the
wrong place. There is no better place to <b>Blip de Bit</b> like
  <i>blipdebit.com</i>.
</p>
<p><small>Blip de Bip is not a registered trademark name.</small></p>
</body>
</html>
```

Figure 2-3 shows the web page, loaded into Firefox.

Earlier I mentioned about closing tags and nested elements, and how the order of the tags should be maintained. This is even more crucial for inline elements. As an example, if you're using both bold and italic markup at the same time, you need to ensure that the order of closing tags follows the order of the opening tags, or you could end up with an unexpected web page display.

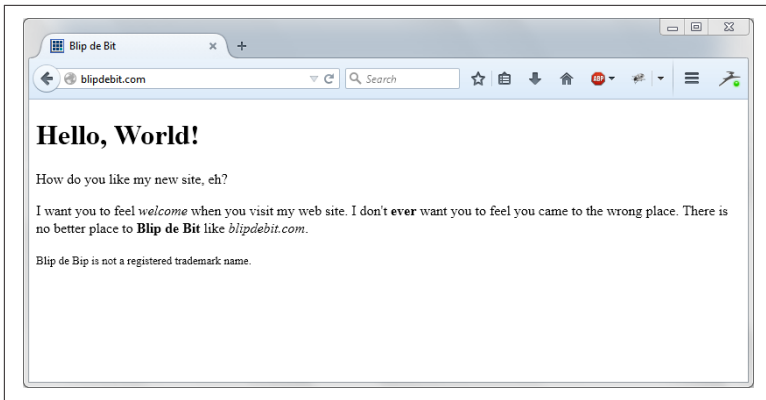


Figure 2-3. *Blip de Bit* web page with inline elements loaded into Firefox

The following demonstrates how *not* to close inline elements:

```
<p>This is <b>a big <i>chunk</b> of text</i>.</p>
```

while a proper use of tags is demonstrated in the following:

```
<p>This is <b>a big <i>chunk</i> of text</b>.</p>
```

Returning to the inline elements, a very common element is an inline element that has characteristics of a block-level element: the **br**, or break, element. It inserts a new line into the web page, and that's all it does:

```
Some text<br />Some other text
```

The web page would show the first part of the web contents on one line, the second part (after the **br**) on another line. It's an empty element, and in HTML5 you can skip the last forward slash if you wish.

Another popular inline element is the **span**; I'll cover it, as well as the ubiquitous **div** element, in the last section in this chapter.

Common Block-Level Elements

Block-level elements are ones that expand to fit the container width, and insert a line break, both before and after the element, in the page flow. Other than these two shared characteristics, most are significantly different from each other.

The paragraph (**p**) is the most common of the block-level elements, and you've seen it demonstrated in both Examples 2-2 and 2-3. Two other common elements frequently found in web pages are the ordered list (**ol**) and the unordered list (**ul**). They're used for listing information, but they're also used to create menus and *drop-down* option lists. The difference between the two is that the order of list items in an unordered list is irrelevant, while the order of the list items in an ordered list are very relevant.

Two lists in a body element are created in the following code to demonstrate how the list contents are displayed in a web page. Note that the list items in both lists are delimited by a third element, the aptly named list item (**li**). The list item can be text, or it can be other HTML elements, including new lists:

```
<body>
<ul>
  <li>Apples</li>
  <li>Oranges</li>
  <li>Bananas</li>
  <li>Berries: <ul>
    <li>Strawberry</li>
    <li>Blueberry</li>
  </ul></li>
  <li>Pineapples</li>
</ul>
<ol>
  <li>First item</li>
  <li>Second item</li>
  <li>Sub-items:
    <ol><li>First sub-item</li>
    <li>Second sub-item</li>
  </ol></li>
  <li>Fourth item</li>
</ol>
</body>
```

Figure 2-4 displays the result of this HTML in a Firefox browser. The **ul** list items are indicated with a closed circle in the outer list, and an open circle in the nested **ul** element. The symbol used with the list items changes for each level of nesting. The **ol** contents are annotated with a number representing their position within their parent container. The nested **ol** items numbering starts over from the beginning. The list items are indented in their parent element (whether **body** or **li**), by default.

- Apples
 - Oranges
 - Bananas
 - Berries:
 - Strawberry
 - Blueberry
 - Pineapples
-
1. First item
 2. Second item
 3. Sub-items:
 1. First sub-item
 2. Second sub-item
 4. Fourth item

Figure 2-4. Unordered and ordered lists and nested lists

Another popular block-level element is **blockquote**. You can use it to embed longer quotations in a web page:

```
<blockquote>
  <p>Another popular block-level element is blockquote. You can
  use it to embed longer quotations in a web page, and include
  an optional citation for the quote.
</p>
</blockquote>
```

The quoted text is embedded in a paragraph element, which is then nested in the **blockquote** element. By default, the **blockquote** contents are indented before being displayed, as shown in [Figure 2-5](#), where the **blockquote** is embedded between two paragraphs and displayed in Chrome.

Following is a blockquote:

Another popular block-level element is blockquote. You can use it to embed longer quotations in a web page, and include an optional citation for the quote.

That was a blockquote.

Figure 2-5. Blockquote demonstrated in Chrome

One element that used to be ubiquitous in web pages before CSS is the **table**. It was ubiquitous because it was a simple way to organize

a web page's structure. Now, its use is strictly for organization and presentation of data.

A simple HTML table consists of table rows (via the **tr** element), which contain the data in cells (enclosed in **td** elements), with column headers (managed with **th** elements). An example of a simple three-column, three-row table is shown in [Example 2-3](#).

Example 2-3. A simple three-column, three-row HTML table

```
<table>
  <tr>
    <th>Column 1</th>
    <th>Column 2</th>
    <th>Column 3</th>
  </tr>
  <tr>
    <td>4.56</td><td>98.12</td><td>100.66</td>
  </tr>
  <tr>
    <td>35.77</td><td>98.01</td><td>3906.04</td>
  </tr>
  <tr>
    <td>45.77</td>
    <td>3.45</td>
    <td>356.99</td>
  </tr>
</table>
```

The default display for this table in IE is shown in [Figure 2-6](#). Note that the **th** contents are in bold text, and that the data in the table is aligned by column and row.

Column 1	Column 2	Column 3
4.56	98.12	100.66
35.77	98.01	3906.04
45.77	3.45	356.99

Figure 2-6. A basic three-column, three-row HTML table

The table has no outline. If you want to give it an outline, and don't want to use CSS, you can do so by using the **border** attribute in the table element:

```
<table border="1">
```

The browser draws a border around the table, and around each table data cell and column header (as shown in [Figure 2-7](#)), after the example is modified with the addition of the **border** attribute and opened in Firefox.

Column 1	Column 2	Column 3
4.56	98.12	100.66
35.77	98.01	3906.04
45.77	3.45	356.99

Figure 2-7. HTML table with the addition of the border attribute

Other common block-level elements are **form** elements, which I'll introduce in [Chapter 4](#), and the **div** element, covered later in its own section. Before leaving the block-level elements, I did want to point out the use of whitespace (the new lines and indentation) in the HTML, and what it means to the page layout.

In the HTML table code, the data cells for two of the table rows are defined on one line, while in the third, the individual **td** elements are separated by new lines. In addition, indentation is used with the nested elements. None of this use of whitespace impacts on the display, because for the most part, whitespace outside of attribute values is treated the same in HTML: it's all equivalent to a single space. If we use three consecutive tabs, they appear as a single space in the web page. If you separate two elements by several new lines, they also appear only as a single space.

The only reason we use the whitespace is that it makes the HTML more readable. Readable HTML is HTML we can more easily modify or correct. However, with so many content management systems (CMS) creating web pages automatically, the HTML for many web pages is compressed—sometimes completely, with most new lines and indentation stripped out.

It doesn't matter to the browser if you use new lines and indentation. It does matter to people trying to edit the HTML.

The div and span Elements

Two elements used in most web pages are the **div** and the **span** element. Their only purpose is to group: either elements (**div**), or text

(**span**). They provide a way to do something with the group without attaching any semantics to the grouping.

Many web pages use the **div** element to mark sections of the web page, such as the sidebar, main content, footers, etc. Before HTML5, the **div** element was the *only* way to mark off sections of the web page for a specific purpose. They provided a way to apply CSS styling (covered in [Chapter 3](#)) to a group of elements as a whole:

```
<div id="sidebar">
  <p>some content</p>
  <p>some other content</p>
</div>
```

The **div** element frequently has either the **class** or **id** attribute, in order to identify the element (or group of elements), for styling.

The same use of attributes and grouping applies to the **span**. The **span** element allows us to provide styling for a block of text without having to use semantic elements (e.g., **strong** or **em**). The **span** element also allows us to provide other, arbitrary styling (provide a background color, change font, etc.).

Without CSS, neither the **span** or **div** element would be that much use, so it's time to leave our very quick introduction to HTML and spend some time with this capability.



Additional Reading

For a more detailed look at HTML, I recommend *Creating a Website: The Missing Manual* by Matthew MacDonald (O'Reilly). Mozilla also provides a [nice introduction to learning HTML](#).

Styling the Page with CSS

Once you've formatted your web page with HTML, you're going to want to liven it up. Though web pages using default styling are functional, they're not that interesting-looking or unique. To make them so, you'll need Cascading Style Sheets (CSS).

CSS is a way of adding visual and other styling to web page contents. It can be applied directly to an element, using the **style** attribute. In the following HTML, the background color for the **div** element contents is set to red:

```
<div style="background-color: red">...</div>
```

Using **style** attributes directly in elements is supported in all of the browsers, but its use is discouraged. Making a style change means having to search for all uses of the **style** attribute and changing them individually. A preferred approach is to use a *stylesheet*.

CSS Stylesheets

A stylesheet is a listing of CSS settings grouped together. By grouping the settings together, we can easily find the style settings to make our changes. The stylesheet can either be embedded directly into the web page or linked in as an external document. The latter approach is particularly favored, because then we can apply the same CSS across several different web pages without having to repeat the same CSS in each page.

To include an embedded stylesheet, add a **style** element to the **head** element in the web page:

```
<head>
  <style>
    ...
  </style>
</head>
```

You can specify a **type** attribute for the element, but the default type of `text/css` is sufficient for our needs.

To link an external CSS stylesheet, using the **link** element:

```
<link type="text/css" rel="stylesheet" media="screen"
href="./css/main.css" />
```

A CSS stylesheet link has a **type** of `text/css`, and the **rel** attribute is set to `stylesheet`. The **rel** attribute establishes the relationship type of the linked document. The **media** attribute specifies the media type to which it applies, such as `screen` in this instance. This setting means the stylesheet is configured for computer screens.

The last attribute in the link is **href**, containing the *relative* or *absolute* URL for the stylesheet. An absolute URL provides the entire URL (e.g., `http://somedomain.com/main.css`), while a relative URL provides information about where the file is located relative to the current document. In the example, the `main.css` file is located in the `css` subdirectory directly accessible from the host page's location. The `./` signals the current web page location. If the host web page is located at the top-level directory of the website, we could also use `/css/main.css`.

Applying CSS

There are several basic ways you can apply CSS to web page elements, and in this chapter, I'm covering the most basic. The most common and simplest is to apply the CSS to an entire set of a specific type of element. As an example, if we want to change the default font family for all paragraph elements in a web page, we'd use the following CSS:

```
p {
  font-family: Times, "Times New Roman", Georgia, serif;
}
```

The selector is the element name, followed by an opening curly bracket (brace), the CSS, and the closing curly bracket/brace. Unless the style setting is overridden later in the same stylesheet, or in

another stylesheet linked after this one in the document, all paragraphs in the page use one of Times, Times New Roman, Roman, Georgia, or serif. The reason multiple values are listed is that not all browsers support all font families. The browser traverses the list and stops once it finds a font family it does support.

The style setting ends with a semicolon. The style *rule* itself consists of a property name (**font-family**) and the setting. Multiple style settings can be given for the same element(s):

```
p {
    font-family: Times, "Times New Roman", Georgia, serif;
    font-size: 14px;
}
```

If we're only interested in applying the style setting to certain paragraphs, we can use the **class** attribute. The **class** attribute is added to the target paragraphs:

```
<p class="highlight">...</p>
```

And the class name is used in the stylesheet, preceded by a dot, indicating that the item is a class name:

```
p.highlight {
    font-family: Times, "Times New Roman", Georgia, serif;
    font-size: 14px;
}
```

If you want the style setting to apply to *all* elements with the given class name, leave off the element designation:

```
.highlight {
    font-family: Times, "Times New Roman", Georgia, serif;
    font-size: 14px;
}
```

If you want to be more selective and apply a style setting to only one element, use the **id** attribute in the element:

```
<div id="sidebar">...</div>
```

Now, the style setting applies only to the one given element:

```
#sidebar {
    width: 200px;
    margin: 10px;
    float: right;
}
```

Many (but not all) CSS styles are *inheritable*, which means when you apply a CSS style setting to an element, it applies to all elements contained within that element. So if you have a **div** element with a couple of paragraphs, setting the **font** family for the **div** element, also sets the font family for the contained paragraphs:

```
#sidebar {  
    font-family: Verdana, Geneva, sans-serif;  
}
```

The only time the paragraphs won't inherit the CSS for the parent element is if it has its own CSS setting:

```
#sidebar p {  
    font-family: Georgia, serif;  
}
```

This stylesheet *selector* applies to all paragraphs that are nested within the **div** element with an **id** of **sidebar**. Any other elements contained in the same **div** element inherit the sans-serif fonts, while the paragraphs have their own serif fonts.



That Cascading Thing

Another impact on the CSS setting for an element is if there are multiple CSS stylesheets, all applying their own rules to the same element.

The browser then uses an algorithm, the CSS *cascade*, to determine which settings are applied to which elements. I defer to my favorite website, Mozilla Developer Network, for a detailed explanation of the [CSS cascade](#).

There is also a way of providing a CSS selector pattern, such as looking for all first paragraphs in a **div** block and so on, but this is beyond the scope of this book. At the end of the chapter, I'll provide references to help you get into more advanced CSS.

The CSS

Now that you know how to add the CSS to the web page, let's dive directly into the most common CSS you'll need or are likely to encounter until you get into more advanced web page design.

To demonstrate the impact various style settings can have on a web page, let's start with an HTML page, shown in [Example 3-1](#), with various elements that we'll style throughout this section.

Example 3-1. Example HTML used to demonstrate style settings

```
<body>
  <div id="main">
    <h1>Hello, World!</h1>
    <p>Now that you know how to add the CSS to the web page,
let's dive directly into the most common CSS you'll need, or
are likely to encounter until you get into more advanced web
page design.</p>
    <p>The most significant change you can make to a web
page is altering the fonts used for all of the text. The
default fonts provided by the browsers are serviceable but
don't provide the unique look and feel you want from your site.</p>
  </div>
  <div id="sidebar">
    <ul>
      <li>First item</li>
      <li>Next item</li>
      <li>Last item in list</li>
    </ul>
  </div>
</body>
```

Without custom CSS styling, the page looks like the one shown in [Figure 3-1](#), when loaded in Opera.



Figure 3-1. Demonstration HTML with default styling

Fonts

The most significant change you can make to a web page is altering the fonts used for all of the text. The default fonts provided by the browsers are serviceable, but don't provide the unique look and feel you want for your site.

The fonts can be changed for every element that displays text. This includes headers and paragraphs, as well as the content in lists. To demonstrate, let's start with a simple stylesheet that changes the font family for the **h1**, **p**, and **li** page elements. We'll mix it up a bit, using different types of fonts:

```
<style>
  h1 {
    font-family: Times, "Times New Roman", serif;
  }
  p {
    font-family: Arial, Helvetica, sans-serif;
  }
  li {
    font-family: "Lucida Console", Monaco, monospace;
  }
</style>
```

I used three different types of font families: serif, sans-serif, and monospace. The results are shown in **Figure 3-2**. As you'll notice, the header isn't different from the default styling, because headers are usually styled using Times New Roman. The paragraphs and list items, though, are significantly different.

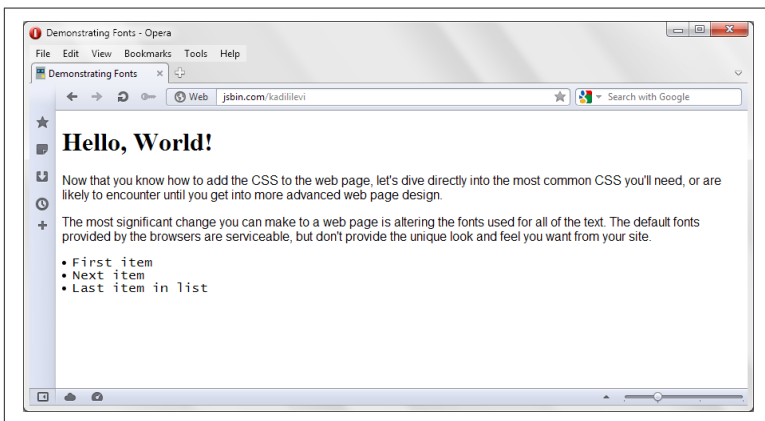


Figure 3-2. Demonstration web page with changed font families

Font families aren't the only thing you can change about fonts. The list of font settings you can style with CSS is comprehensive:

- **font-weight**
- **font-style**
- **font-variant**
- **font-family**
- **font-size**
- **line-height**

I include **line-height** in the list because of its impact on the overall appearance of the font.

Let's take a look at how each of these properties impacts the font. The **font-weight** property sets the font's weight—making the font appear as normal to progressively heavier and more bold. You can specify a font weight number (e.g., 400) or keyword (e.g., bold), return the font weight to its default value (`initial`), or `inherit` the **font-weight** setting from any parent element's CSS settings.

Let's do something different. Let's make the header lighter and the font in the list bolder. We'll leave the paragraph text alone:

```
<style>
  h1 {
    font-family: Times, "Times New Roman", serif;
    font-weight: normal;
  }
  p {
    font-family: Arial, Helvetica, sans-serif;
  }
  li {
    font-family: "Lucida Console", Monaco, monospace;
    font-weight: 900;
  }
</style>
```

The impact on the header isn't as noticeable, but the impact on the list items is obvious, as shown in [Figure 3-3](#).

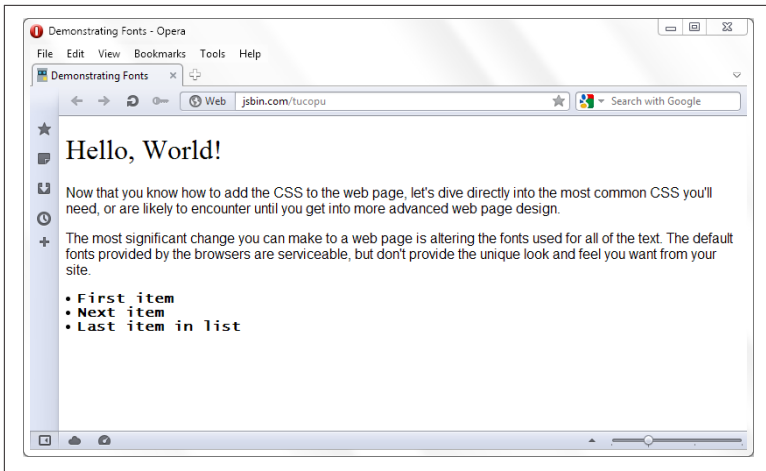


Figure 3-3. Example after modifying font-weight for header and list

The reason the change with the header isn't as apparent is its large size. We can change that, though, just by changing the font size of the header, using the **font-size** property:

```
h1 {  
    font-family: Times, "Times New Roman", serif;  
    font-weight: normal;  
    font-size: 1em;  
}
```

The **font-size** property supports multiple types of sizing units, including the *em* units in the code, as well as pixels, expressed as *px*. One *em* unit is equivalent to the font size of the parent element, or if not specified, equal to the default font size, which is 16 pixels. Because of this sizing algorithm, *em* units are dynamic. If you want to use specific sizes, then use pixels:

```
font-size: 12px;
```

You can also use enumerated values, like **small**, **medium**, and **large**; or relative size, like **smaller** and **larger**. Another approach is to use a percentage:

```
font-size: 50%;
```

If you're only interested in modifying some text within an element, you can utilize the **span** element. I modified the first paragraph to include a span element with an **id** attribute:

```
<p>Now that you know how to add the CSS to the web page,  
<span id="text1">let's dive directly into the most common  
CSS you'll need</span>, or are likely to encounter until you  
get into more advanced web page design.</p>
```

Now I'm going to modify the **font-style** property for this **span**:

```
#text1 {  
    font-style: oblique;  
}
```

The text enclosed in the **span** is now oblique, a variation on the italic style. But it's still not enough. Let's also modify the text's **font-variant**, setting it to **small-caps**:

```
#text1 {  
    font-style: oblique;  
    font-variant: small-caps;  
}
```

The last property I'm covering in this section is **line-height**. It's not a font property but can influence the appearance of the text. To make text more readable, adjust the **line-height**.

To demonstrate, add another **id** attribute, this time to the second paragraph:

```
<p id="second">...</p>
```

and then add styling to decrease the **font-size** and increase the **line-height**:

```
#second {  
    font-size: smaller;  
    line-height: 150%;  
}
```

The smaller font size could make the text harder to read, but the increased line height adds clarity, as shown in [Figure 3-4](#).

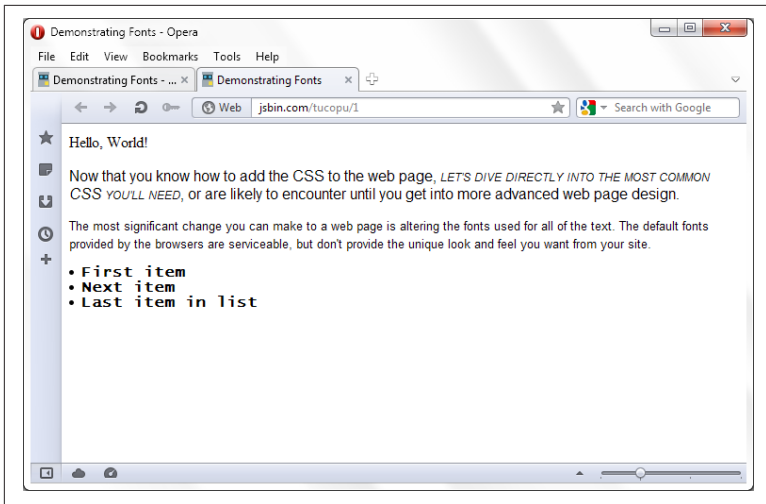


Figure 3-4. Web page after all the font modifications

The font properties are handy, but sometimes it gets cumbersome having to list all of them. The last font property is **font** itself. You can use it to change several font properties, all in one setting. To demonstrate, let's change the style setting for the second paragraph, using **font** to set several properties at once:

```
#second {  
  font: italic bold 14px/36px "Comic Sans MS", cursive,  
    serif;  
}
```

The properties changed are **font-style**, **font-weight**, **font-size/line-height**, and **font-family**. Now take a look at the web page, in [Figure 3-5](#). It's definitely unique!

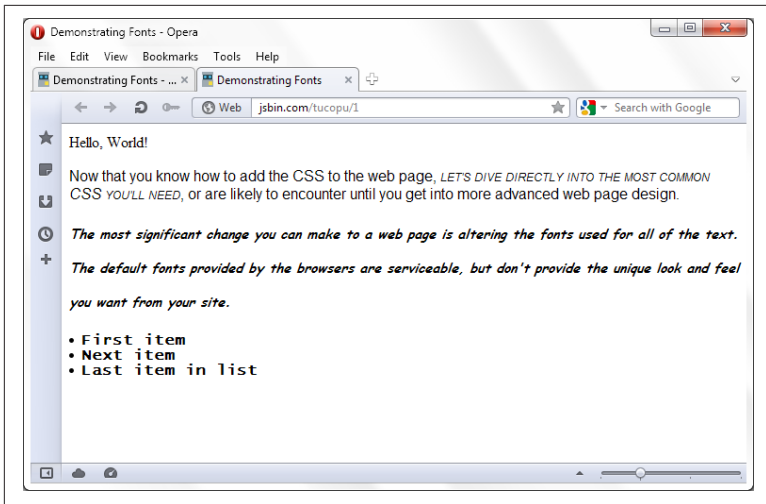


Figure 3-5. Going absolutely mad with the font CSS properties

Decorative Accents and Color

One aspect of the font we didn't change in the last section is the font's color. The CSS property to do so is **color**:

```
#second {  
  font: italic bold 14px/36px "Comic Sans MS", cursive,  
    sans-serif;  
  color: red;  
}
```

If you want to get particularly colorful, you can also change the background color of the element's contents:

```
#second {  
  font: italic bold 14px/36px "Comic Sans MS", cursive,  
    sans-serif;  
  color: red;  
  background-color: #ffff00;  
}
```

Both examples demonstrate two ways to specify a color in CSS. The first uses a color name, from the **many that are supported**. The second uses a hexadecimal color. A third approach is to use RGB notation:

```
background-color: rgb(255,255,0);
```

RGB stands for red, green, and blue, the three colors that, combined, make up all of the colors we see in web pages. Each color

value has a range from zero to 255, with 255 representing the strongest saturation of color. The hexadecimal notation is a conversion of the digital value (0...255) into hexadecimal, with a value of 255 equivalent to *ff*. The hexadecimal version is a combination of the colors without the parentheses, and use of **rgb** annotation. So the following are both equivalent:

```
background-color: rgb(255,255,0);  
background-color: #ffff00;
```

You can also use a shorthand version of the hexadecimal number. The hexadecimal number for yellow can be *#ffff00*, or it can be given as *#ff0*. The individual two-character representation for each color value can be shortened to one, if both characters for each value are the same.

If a color isn't enough for the background, you can also use an image as wallpaper for the element:

```
#header {  
  background-image: url(myimage.jpg);  
  background-size: 100%;  
}
```

The image given for the background is sized to fit the element's width, with the height adjusted automatically to compensate. Other background image properties are:

- **background-repeat:** Repeat along y- or x-axis, or both
- **background-position:** Where to position background image in element
- **background-clip:** How to clip an image larger than element
- **background-origin:** Position of origin of image
- **background-attachment:** Whether image is fixed or scrolls with the element as user scrolls

Just specifying an image sets other values to their default:

- The image's initial size is used.
- The image's upper-left corner is placed in the element's upper-left corner.
- The image is repeated, across and down, until it fills the container.
- The image scrolls with the element.

Let's add some color to the example page. First, we'll modify the HTML to add a **div** element wrapper around the **h1** header, and add a new paragraph with author name:

```
<div id="header">
  <h1>Hello, World!</h1>
  <p>Shelley Powers</p>
</div>
```

Next, we'll adjust the font for the paragraph given the **id** value of second, because we really don't want to use Comic Sans MS in our web pages. (Whether deserved or not, Comic Sans MS is almost universally loathed by a surprisingly large number of people.)

The line spacing was good for demonstration, but now we want to clean it up, too, so the paragraph looks like we're serious about design:

```
#second {
  font: 14px/20px "Georgia", serif;
}
```

Lastly, in preparation, we'll change the **h1** font size back to its default by removing all of the style settings for it. All the changes leave us with the web page in [Example 3-2](#).

Example 3-2. Example page in preparation for adding some color

```
<!doctype html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>Blip de Bit</title>
  <meta name="description" content="Blip de Bit!">
<style>
  h1 {
    font-family: Times, "Times New Roman", serif;
    font-weight: normal;
    font-size: 1em;
  }
  p {
    font-family: Arial, Helvetica, sans-serif;
  }
  li {
    font-family: "Lucida Console", Monaco, monospace;
    font-weight: 900;
  }
  #text1 {
    font-style: oblique;
    font-variant: small-caps;
  }
```

```

    }
    #second {
        font: 14px/20px "Georgia", serif;
    }
</style>
</head>
<body>
    <div id="main">
        <div id="header">
            <h1>Hello, World!</h1>
            <p>Shelley Powers</p>
        </div>
        <p>Now that you know how to add the CSS to the web page,
        <span id="text1">let's dive directly into the most common CSS you'll
        need</span>, or are likely to encounter until you get into more
        advanced web page design.</p>
        <p id="second">The most significant change you can make to a
        web page is altering the fonts used for all of the text. The default
        fonts provided by the browsers are serviceable, but don't provide
        the unique look and feel you want from your site.</p>
        </div>
        <div id="sidebar">
            <ul>
                <li>First item</li>
                <li>Next item</li>
                <li>Last item in list</li>
            </ul>
        </div>
    </body>
</html>

```

Now we're ready to paint the page. First, we'll add a background image for the header. It's a nice generalized darker background, and I'll set it to repeat along the x-axis. Since we're using a darker background, we'll change the font color within the header to white:

```

#header {
    background-image: url(background.jpg);
    background-repeat: repeat-x;
    color: white;
}

```

Next, we'll add some colorful highlights to the inline **span** text:

```

#text1 {
    font-style: oblique;
    font-variant: small-caps;
    color: rgb(255,0,0);
    background-color: #ffff00;
}

```

With just these two pages, the web page has come alive, as shown in [Figure 3-6](#).

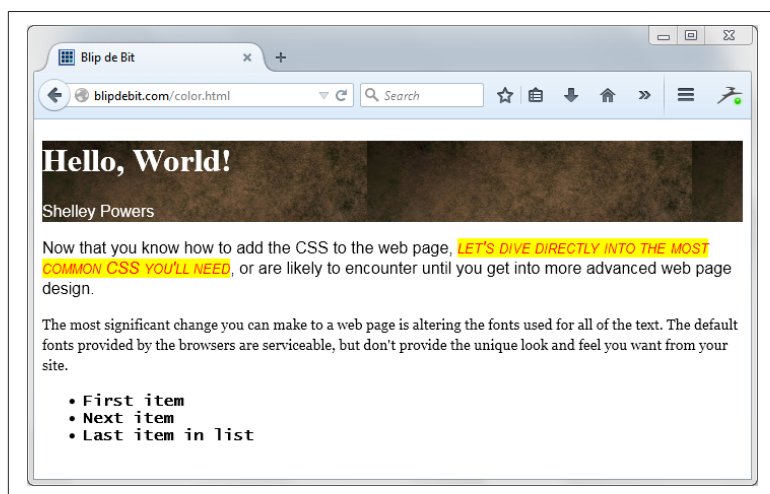


Figure 3-6. Adding a little color to fonts and background

This is an improvement. The highlighted text is much more noticeable, and the header more distinct. We're not finished, though.

The sidebar listing can be improved. That's the group of list items at the bottom. The first change we'll make is to add a nice border around the element, using the CSS border properties. The border properties can either be used to add the same border to all sides of the element's container, or each side can be different. And each border can have a different color, width, and design, by altering the following CSS properties:

- **border**
- **border-width**
- **border-style**
- **border-color**

There are other border properties, but these are the most commonly applied. The **border** property is a shorthand property, which can be used to set the **border-width**, **border-color**, and **border-style**, all at once. So we can define a border that's orange, three pixels wide, and solid by modifying the individual properties:

```
#sidebar {  
    border-width: 3px;
```

```
border-style: solid;
border-color: orange;
}
```

Or we can declare all three at one time:

```
#sidebar {
  border: 3x solid orange;
}
```

I like simplicity, so we'll go with the latter approach. We can also define the border differently for all the sides by using the **border-bottom**-, **border-top**-, **border-left**-, and **border-right**- annotation. It's not uncommon for the top and bottom borders of a sidebar to be slightly different than the sides. So we'll use the following for the sidebar element (pulling in more **named web colors**):

```
#sidebar {
  border-left: 1px solid PaleGoldenRod;
  border-right: 1px solid PaleGoldenRod;
  border-bottom: 3px solid orange;
  border-top: 3px solid orange;
}
```

We're not done with the sidebar, yet. The list items will eventually be turned into links, which means the list style annotation is going to look a little odd. We want to get rid of the list item default circles and just leave the text.

The decoration for the list item elements is controlled by a set of properties:

- **list-style-type**
- **list-style-position**
- **list-style-image**

By default, the **list-style-type** is a disc, as displayed in previous figures. Other values include **circle**, **square**, **decimal**, and so on. If the built-in styles aren't sufficient, you can supply an image to use via **list-style-image**. You can also specify if the image is displayed within the block (value of **inside**) for the list item or outside the block (value of **outside**). By default, the image is displayed outside the block.

All three values can be set using the shorthand **list-style** property:

```
ul {
```

```
list-style: square inside none;
}
```

This CSS changes the list items for the given **ul** element to a square image, inside the block, and no user-supplied image.

The list style properties are an interesting variation in CSS: they're defined for the parent **ul** element, though displayed for each **li** element. To make the change we want to make to the sidebar—remove any decoration on the individual list items—the **list-style-type** property for the parent element is set to a value of none:

```
#sidebar ul {
  list-style-type: none;
}
```

The CSS demonstrates how you can attach a style to an element that's a child of another element. The **div** element has an **id**, but the **ul** element doesn't. We don't want to remove the decoration for all lists in the page, so we specify that we want to attach the style only to unordered lists that are children of the **div** element with the **id** of sidebar.

Figure 3-7 shows the page with the new modification. It's starting to look a little more polished, but there are still some tweaks left to apply.

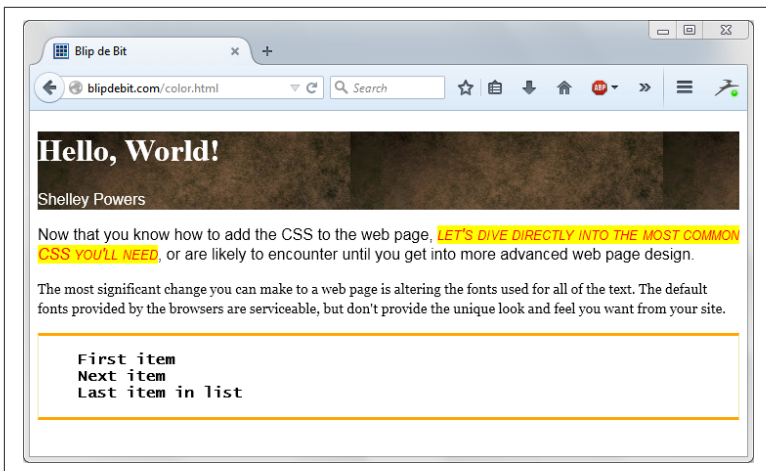


Figure 3-7. Adding border and removing list item decoration

Spacing

Another set of CSS properties that can have significant impact on the web page display is those controlling space, both within and outside an element's boundary box.

The most commonly used of the properties are those related to margins:

- **margin**
- **margin-left**
- **margin-right**
- **margin-top**
- **margin-bottom**

The margin creates space around a web page element. As you can deduce from the property names, you can control the margin for a particular side of the element, more than one side, or for all sides. If you have a **div** element with an **id** of `main`, and want to set the right margin to 10 pixels, you'd use the following CSS:

```
#main {  
    margin-right: 10px;  
}
```

If you're interested in setting all sides to 10 pixels, use the following:

```
#main {  
    margin: 10px;  
}
```

If you want to set each margin to a different value, you could set each margin individually:

```
#main {  
    margin-left: 10px;  
    margin-right: 5px;  
    margin-top: 15px;  
    margin-bottom: 20px;  
}
```

Or set all sides using **margin**:

```
#main {  
    margin: 15px 5px 20px 10px;  
}
```

The shorthand approach sets the margins for the top, right, bottom, and left, in that order. You can also specify only two values:

```
margin: 10px 20px;
```

For setting both the top and bottom margins to the same value (10px), as well as the left and right (20px).

The margin provides spacing outside the element. Each block-level element exists in a box, and the margin creates whitespace around this box. In the example that we're modifying, the page seems to have whitespace between the top and side of the browser's window, and the header. If we want to ensure the header is flush against the top and side of the window, we'll need to make a couple of new settings using margin.

First, we'll need to set the header's **h1** element's **margin-top** to zero. This ensures it's not pushing out against the top of the header:

```
#header h1 {  
    margin-top: 0;  
}
```

Next, we need to make sure the **body** element's own default margin is zeroed out:

```
body {  
    margin: 0;  
}
```

This leads us to one of the little challenges we'll run into from time to time with CSS: the default settings for many of the elements, specifically the margin and padding settings (I'll cover padding a little later). They can play havoc with a web page design, because the overall effect is our CSS *in addition to* the default values. The preferred way to ensure that the default values are not messing around with our carefully wrought design is to use the universal selector (*) and set all the margins and padding values to zero:

```
* { margin: 0px; padding: 0px; }
```

This line is added to the top of the first stylesheet used in the page and helps to eliminate one frustrating problem. The universal selector is a way of applying a CSS setting to *every* element in the page.

Of course, doing this creates its own problems. We'll quickly find out we've lost the whitespace around our paragraphs, our lists, and other elements. Rather than have to add back the necessary whitespace we need for all elements, for now we'll just add the modifications for the **h1** and **body** elements and experiment around with the universal selector another time.

The padding properties operate in a similar manner to the margin, in that you can specify padding for each element's side individually or as a group:

- **padding**
- **padding-left**
- **padding-right**
- **padding-top**
- **padding-bottom**

The padding properties add whitespace *within* the element's box, rather than the outside. Let's look at that header again. It's nicely flush against the browser window, but the elements in it are smashed up against the top and side. What we want to do is give them a little space.

Rather than modify their margins, what we're going to do is add whitespace inside of the header, by modifying the **padding** for the element:

```
#header{
  background-image: url(background.jpg);
  background-repeat: repeat-x;
  color: white;
  padding: 10px 0 10px 10px;
}
```

We're setting the left, top, and bottom of the header's padding values to 10 pixels, and the right to zero.

We're not quite done with margins and padding. The paragraphs now are smooshed up against the left side of the browser's window. We can fix this problem in one of two ways. The first is add a left and right margin for all paragraphs:

```
p {
  margin-left: 10px;
  margin-right: 10px;
}
```

Another approach is to make one modification to the page: pull the header element out of the **main** **div** block, so that the **main** **div** block only wraps the two paragraphs:

```
<div id="header">
  <h1>Hello, World!</h1>
  <p>Shelley Powers</p>
</div>
```



```

<div id="main">
  <p>...</p>
  <p>...</p>
</div>

```

The reason is that now we can set the margin's of the `main` element, and everything contained in `main` benefits from the side whitespace, as shown in [Figure 3-8](#).

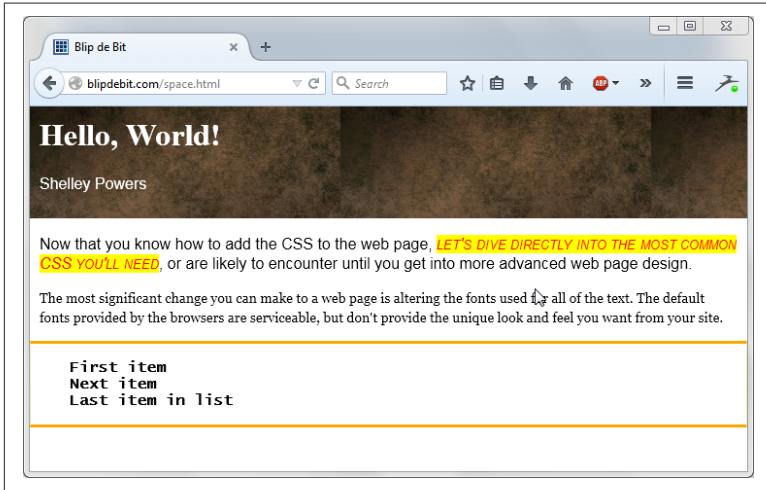


Figure 3-8. After margin and padding modifications

Margins, padding, borders, and the content are all components of the **CSS Box Model**. [Figure 3-9](#) displays the CSS Box Model graphic in the W3C documentation.

Each element has a box around it with the element's contents. Around that is another box representing the element's padding. If the padding is set to zero, the edges of this box are the same as the edges for the content box. Around the padding box is one representing the element's border. If it's set to zero (no border), then the edges of this box are the same as the padding edges.

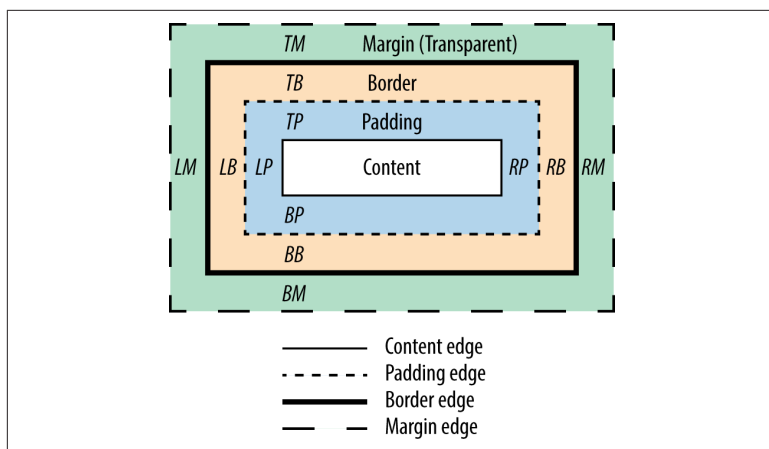


Figure 3-9. CSS Box Model graphic from the W3C CSS documentation

The last box represents the margin. If it's set to zero, its edge would be the same as the border's edge. If the padding, border, and margins of an element are set to zero, then the content edge is what defines the element's width and height. And all of this impacts on the element's position within the page layout, covered next.

Layout

Our page is starting to come together. The only problem is, the sidebar isn't positioned as a sidebar: it's positioned like a footer. We really want to position the main web page contents to one side of the page and the sidebar to the other, with the header along the top. The last modifications we'll make to the example page is to modify the layout of the page so the main and sidebar sections are side by side.

First, we'll have to resize both of these container elements so they'll fill the page when placed side by side. Next, we need to change how they relate to each other so they'll display correctly. The CSS properties to make these changes are the **width** and the **float**. Along the way, we'll also have to tweak margins and padding.

The **width** allows us to control the width of a block-level element, like our `main` and `sidebar` `div` elements. A popular page width is 960 pixels, so we'll set the width of the main element to 650 pixels and the sidebar to 220 pixels. This is a good size for readable text in the `main` element and allows for whitespace around both:

```
#main {
  width: 650px;
}

#sidebar {
  border-left: 1px solid PaleGoldenRod;
  border-right: 1px solid PaleGoldenRod;
  border-bottom: 3px solid orange;
  border-top: 3px solid orange;
  width: 220px;
}
```

The **float** allows the two block elements that would normally stack on each other, to exist side by side. The **float** removes the element from the standard flow, positioning it in either the left or right side of its container, depending on the **float** value. If you set the **float** of two consecutive block-level elements, they'll end up side by side within their parent container. If you set the **float** value to **right**, the first element in the page will float to the right of the first. If you set the **float** value to **left**, the first element will float to the left. We want the second element, the sidebar, to float to the right of the primary main element, so we'll set the **float** value for each to **left**.

Opening the page, we can see both elements are positioned correctly, if too close to each other. However, if the page isn't expanded to accommodate the width of both, then the sidebar pushes down the page and ends up underneath the main content. The float properties position both elements within their container element, which is the **body** element. If the browser window is resized, the **body** element's width is changed, impacting **main** and **sidebar**.

To provide better control of the elements' positions, we'll add another **div** element, given an **id** of **container**. It wraps both the **main** and **sidebar** elements, but not the header. The container element is given a width of 960 pixels, a padding of zero, and the top is set to a value where it won't cover any of the top banner graphic. The side margins are set to **auto**, which means they expand and contract automatically as the browser window is resized. Using the **auto** setting ensures the page contents always stay in the center of the window:

```
#container {
  width: 960px;
  padding: 0;
```

```

    margin: 100px auto 0 auto;
}

```

The main and sidebar elements are given margins to provide whitespace around both:

```

#main {
    margin: 0 10px;
    float: left;
    width: 650px;
}
#sidebar {
    border-left: 1px solid PaleGoldenRod;
    border-right: 1px solid PaleGoldenRod;
    border-bottom: 3px solid orange;
    border-top: 3px solid orange;
    float: left;
    width: 220px;
    margin: 0 0 0 30px;
}

```

We can leave the header fixed to the left regardless of browser size. However, to ensure it's always positioned over the content, it's also given a width of 960 pixels, and its left-right margins are set to auto:

```

#header{
    color: white;
    padding: 10px 0 10px 10px;
    width: 960px; margin: 0 auto;
}

```

One last necessary change: if both of the child elements in the container are to float, then the parent element essentially collapses. To prevent this, we'll need to add one more element: another **div** element with an **id** of footer and a CSS setting with the following:

```

#footer {
    clear: both;
}

```

The element is placed last in the container element. When an element's **clear** property is set, floating elements aren't allowed around it. If it's set to **left**, no floating element is allowed to the left. If it's set to **right**, no floating element is allowed to the right. Setting it to **both** doesn't allow floating elements around it, at all, which means it's pushed down below the floating elements. This serves to ensure the container properly flows around the new element and the two floating elements. We're giving the new element a footer **id** so that

we can find it easily if we want to add footer content at some future time.

Now our page reflects the proper layout, as shown in [Figure 3-10](#).

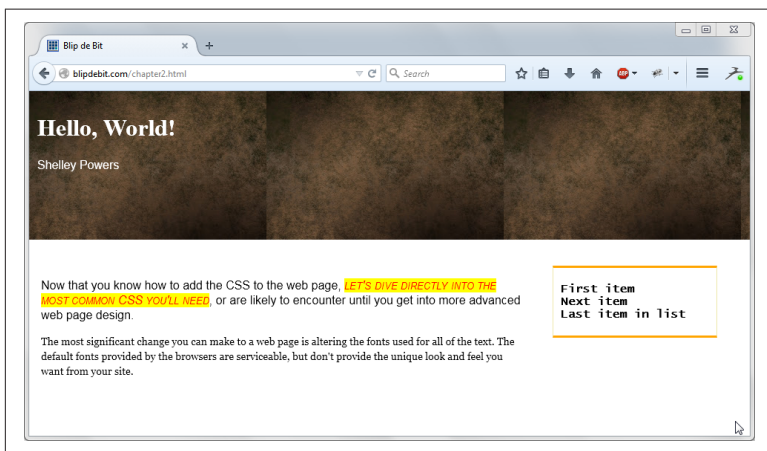


Figure 3-10. Example page with proper layout

Finishing Up

The CSS stylesheet for the example is quite large now. There's far too much CSS to copy and paste from page to page. And it makes the web page harder to read.

To finish the example, we're going to copy the CSS into its own file, named *main.css* (CSS files are given a *cssfile* extension). We're only copying the CSS, not the outer **style** opening and closing tags. The complete contents are given in [Example 3-3](#).

Example 3-3. The CSS stylesheet

```
body {  
    margin: 0;  
    background-image: url(background.jpg);  
    background-repeat: repeat-x;  
}  
#header{  
    color: white;  
    padding: 10px 0 10px 10px;  
    width: 960px;  
    margin: 0 auto;  
}
```

```

#header h1 {
    margin-top: 20px;
}
p {
    font-family: Arial, Helvetica, sans-serif;
}
li {
    font-family: "Lucida Console", Monaco, monospace;
    font-weight: 900;
}
#main {
    margin: 0 10px;
    float: left;
    width: 650px; }
#text1 {
    font-style: oblique;
    font-variant: small-caps;
    color: rgb(255,0,0);
    background-color: #ffff00;
}
#second {
    font: 14px/20px "Georgia", serif;
}
#sidebar {
    border-left: 1px solid PaleGoldenRod;
    border-right: 1px solid PaleGoldenRod;
    border-bottom: 3px solid orange;
    border-top: 3px solid orange;
    float: left;
    width: 220px;
    margin: 0 0 0 30px;
}
#sidebar ul {
    list-style-type: none;
    margin: 20px 10px;
    padding: 0;
}
#container {
    width: 960px;
    padding: 0;
    margin: 100px auto 0 auto;
}
#footer {
    clear: both;
}

```

Once the file is saved, delete the **style** element and its contents from the web page.

To include the newly created external stylesheet, add the following **link** to the **head** element, where the **style** element was originally located:

```
<link type="text/css" rel="stylesheet" href="/main.css"
media="all" />
```

The HTML page is now much cleaner, and you can use the same stylesheet in other pages:

```
<!doctype html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>Blip de Bit</title>
  <meta name="description" content="Blip de Bit!">
  <link type="text/css" rel="stylesheet" href="/main.css"
media="all" />
</head>
<body>
  <div id="header">
    <h1>Hello, World!</h1>
    <p>Shelley Powers</p>
  </div>
  <div id="container">
    <div id="main">
      <p>Now that you know how to add the CSS to the web page,
<span id="text1">let's dive directly into the most common CSS
you'll need</span>, or are likely to encounter until you get
into more advanced web page design.</p> <p id="second">The most
significant change you can make to a web page is altering the
fonts used for all of the text. The default fonts provided by
the browsers are serviceable, but don't provide the unique look
and feel you want from your site.</p>
    </div>
    <div id="sidebar">
      <ul>
        <li>First item</li>
        <li>Next item</li>
        <li>Last item in list</li>
      </ul>
    </div>
    <div id="footer"> </div>
  </div>
</body>
</html>
```

You'll want to test your HTML and CSS in as many browsers as possible, of course, but first you're going to want to check to see if your

use of HTML and CSS is valid. Browsers are forgiving, but each is forgiving in different ways. Just because the page looks good in two browsers doesn't mean it will look good in all.

JavaScript

Interactive functionality in the web page is managed with JavaScript. I'm not introducing you to JavaScript in this publication, but you may find that the template or theme you pick makes use of JavaScript. If so, the files necessary are also included as links within the **head** element.

You don't have to be proficient with JavaScript to utilize it in your site. All you need to do is add links to JavaScript libraries if your theme or template requires it, or if you're using a plugin (covered in [Chapter 5](#)) that's dependent on JavaScript. Templates and themes most likely already include the link, and you'll just need to ensure that the JavaScript is uploaded to your server.

Plugins may require that you add the link yourself, but they'll usually provide instructions in how to do so. Just in case, to add a link to a JavaScript file, use text like the following:

```
<script src="http://yourdomain.com/somejavascript.js">
</script>
```

or the following:

```
<script type="text/javascript" src="/libraries/tool.js">
</script>
```

HTML can be validated in the [W3C's validator](#). You can copy and paste your HTML or just type in the URL for the page you're testing. The validator will check the markup based on the version of HTML specified by the **doctype**. Any errors or warnings are displayed when the validator is finished. If the HTML is valid, you'll get a valid result. Since you're validating HTML5, you'll get one warning, about the fact that the validator is using an experimental HTML5 validator. You can disregard this warning.

To validate your CSS, the W3C provides a [CSS Validator](#). It works the same as the HTML Validator.

HTML and CSS are changing all the time. A great place to check whether a new HTML element or CSS property can be safely used in most browsers is the [Can I use website](#). You can type in the element or CSS property, and the site provides a graph showing browser support. For instance, [Figure 3-11](#) shows the result when I checked for support for the newer input element type of color.

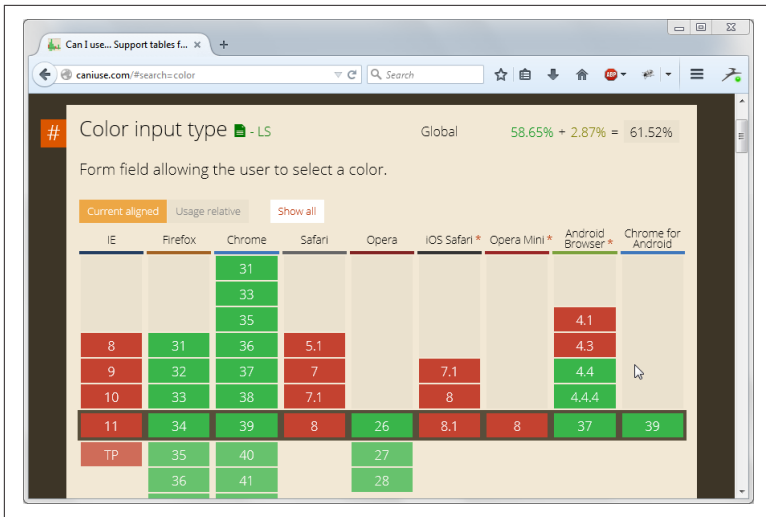


Figure 3-11. Can I Use results for color



For More on CSS

The Mozilla Developer Network provides an excellent **reference for HTML** and the **same for CSS**. When you're ready to get into more advanced CSS, I recommend *CSS3: The Missing Manual* by David Sawyer McFarland (O'Reilly).

Going Beyond the Basics

Now that you've had a chance to work with some basic CSS and HTML, let's get into some more advanced web page construction topics and design: using the new semantic HTML5 elements, working with web forms, incorporating multimedia, and utilizing pre-made design templates.

The New Accessible/Semantic HTML5 Elements

Chapter 2 introduced the **div** element, a versatile element that, until recently, has been the layout and grouping tool of choice in most web page designs. What the **div** element does not provide, however, is semantics. A web page filled with **div** elements is basically a web page filled with undifferentiated blocks of formatted text.

This changed with HTML5, and the introduction of new elements that not only give us the ability to design the page's layout, but also add both semantics and accessibility for the page contents. I'll introduce the elements, and then we'll look at some examples.

The article, section, header, and footer Elements

The first element we'll look at is **article**. It's self-explanatory: an article is a completely contained piece of writing that can be extracted independent of the context. It's equivalent to newspaper and magazine articles, weblog posts, or an entire how-to explanation. This chapter could be considered an article.

The next new element is the **section**. A section is inherently a component of an **article**. It's a block of text subordinate to the overall content of its parent element, but with enough independence to exist as its own entity. If the article is a topic, the sections are sub-topics. Each of the components of this chapter delimited by titled content are equivalent to a section.

The Confusing Semantics of Article and Section

The semantics of **article** and **section** aren't as clearly defined as we could wish. For instance, I consider a book chapter to be equivalent to an **article**, and each component, a **section**. The W3C specification considers the entire book to be an **article**, and each chapter to be a **section**. However, this definition leaves no room to meaningfully categorize all the components of each chapter.

If we consider that the body of the book is equivalent to the **body** of a web page, then my interpretation should be the most accurate. However, others would disagree.

The use of **article** and **section** in web pages seems more clear cut: each discrete piece of content is an **article**, and its components are **sections**. Except that a **section** can also be used as a grouping mechanism for several distinct **articles**. However, using a **section** as a purely grouping element, to me, weakens its semantic meaning.

When it comes to grouping articles, I recommend using the new **main** element. It can only be used once per web page and is a natural for grouping articles. And I'd resist too much nesting of **articles** in **sections** in **articles**, and so on.

Regardless of the approach you take, the most important consideration when incorporating these elements into your layout design is to use them consistently.

In both the **article** and **section**, the heading for each (if one is needed) is defined by the appropriately named **header** element. If there's any ancillary information, for a **section** or an article, it can be included in a **footer** element.

Let's see how these elements hold together. In [Example 4-1](#), a web page representing a typical weblog post is created using the new HTML5 semantic elements. The section elements are nested within the article, and a header element is used to group together the open-

ing **h2** and **p** elements for the article. One of the section elements has a **footer**, as does the **article**.

Example 4-1. Using the new HTML5 elements for a weblog post

```
<!doctype html><html lang="en">
<head>
  <meta charset="utf-8">
  <title>Blip de Bit</title>
  <meta name="description" content="Blip de Bit!">
</head>
<body>
<h1>Welcome to my fantastic weblog</h1>
<article>
  <header>
    <h2>A Typical Weblog Post</h2>
    <p>Author: Shelley Powers</p>
  </header>
  <p>Writing about stuff.</p>
  <section>
    <h3>Good stuff</h3>
    <p>This is the good stuff. Kittens and unicorns.</p>
    <footer>
      <p>More about the good stuff...</p>
    </footer>
  </section>
  <section>
    <h3>Bad stuff</h3>
    <p>And now the bad stuff. Daleks and Weeping Angels.</p>
  </section>
  <footer>
    <p>Additional information about stuff in general.</p>
  </footer>
</article>
</body>
</html>
```

As with all HTML5, none of the closing tags for the new elements are required, but I think they help keep the contents organized and make the document easier to read and modify.

Figure 4-1 shows the web page. As you can see, the new elements don't have a default display different than the previously used **div** elements. Like the **div** element, they're block-level elements, meaning their content expands to fit the width of the container element, and a new line is inserted in the web page just before their position.

The only purpose for the new elements is to add semantics to the web page markup, making it easier for search engines and other automated tools to correctly process the contents. If you want a unique display for each type of element, you'll need to use CSS.

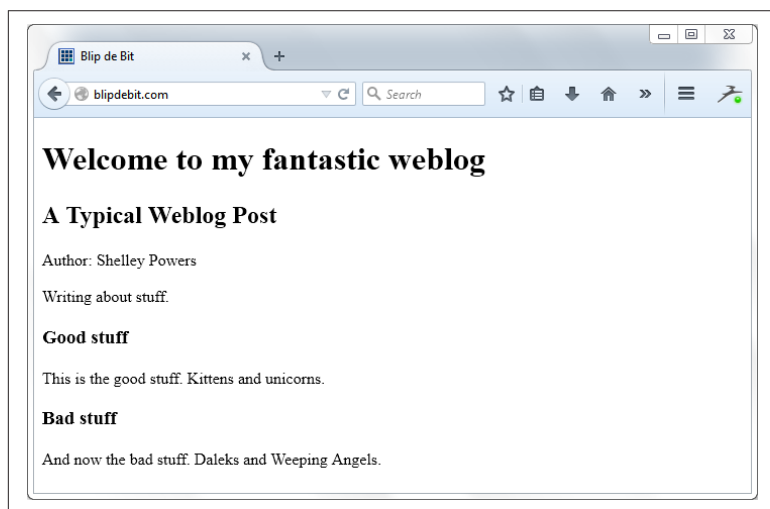


Figure 4-1. A web page using the new HTML5 article, section, header, and footer elements

The new elements also make the page more accessible. Screen readers need assistance in determining which content should be read, and in what order. We can use a specialized set of attributes and markup, known as the **Accessible Rich Internet Applications (WAI-ARIA)** to mark the HTML, but doing so adds an additional level of work. If we use the new HTML5 semantic elements, the necessary accessibility indicators are incorporated by default.

The nav, aside, and figure Elements

Three other new elements work with the ones just covered, to provide other semantically meaningful annotation of the web page content.

The new **nav** element is used to indicate navigation and is a natural for wrapping around navigational menus created using an unordered list (**ul**):

```

<nav>
  <ul>
    <li><a href="/about.html">About Blip de Bit</a></li>
    <li><a href="/contact.html">Contact Blip de Bit</a></li>
    <li><a href="/shop.html">Shop for great stuff</a></li>
  </ul>
</nav>

```

The **nav** element isn't used with any set of links: it's intended purpose is to wrap links used for the website's internal navigation.

The **aside** element is an oddly named element primarily because the more appropriate name, *sidebar*, is already used in web page terminology. The **aside** is equivalent to the sidebar that we find in books —*not* the sidebar located in the side of web pages and containing links to recent stories, ads, etc. The **aside** element can be used for *pull quotes*, for digressions related to an article topic, yet not essential to the topic, or for any other use similar to how sidebars are used in other publications.

Use the **aside** as you would normally use a **div** element:

```

<p>Other material on topic.</p>
<aside>
  <h2>Optional header if appropriate</h2>
  <p>Whatever is the text of the aside.</p>
</aside>
<p>Continuing with topic material, as if the aside
didn't happen.</p>

```

If you want the material in the **aside** to stand out in some way, you'll need to modify the contents with CSS. A popular approach for pull quotes is to have the material float to the left or right and appear in larger font.

Figure 4-2 displays a web page with contents generated using a **Lorem Ipsum generator**. Lorem Ipsum is dummy text used in the print industry as filler when testing new type. It's been used since the 1500s, and the web industry has gleefully adopted it as our own. The page also uses a **nav** element, for navigation, as well as **article**, **header**, **footer**, and **section**. Lastly, it makes use of the **aside** as a pull quote.


```

        background-color: yellow;
    }
    article {
        margin: 20px;
    }
    #first {
        float: left;
        width: 300px;
        padding: 10px;
        margin-right: 10px;
        color: gray;
        border: 1px solid black;
        font-size: larger;
    }
</style>
</head>
<body>
<nav>
    <ul>
        <li><a href="/about.html">About Blip de Bit</a></li>
        <li><a href="/contact.html">Contact Blip de Bit</a></li>
        <li><a href="/shop.html">Shop for great stuff</a></li>
    </ul>
</nav>
<h1>Welcome to my fantastic weblog!</h1>
<article>
    <header>
        <h2>A Typical Weblog Post</h2>
        <p>Author: Shelley Powers</p>
    </header>
    <section>
        <p>    Lorem ipsum dolor...    </p>
        <aside id="first">
            <h2>Pay attention</h2>
            <p>    Vestibulum dictum gravida magna...    </p>
        </aside>
        <p>    Cras at dictum lectus...    </p>
        <p>    Nullam dignissim ligula eu ...    </p>
    </section>
</article>
</body>
</html>

```

The **figure** element and its related **figcaption** provide a long-needed method of pairing captions with figures. The **img** element allows us to embed images in the web page, but not associate a caption directly with that image. The **figure** and **figcaption** elements work with **img** to correct this missing markup.

You still use the **img** element, as you always have, but now you wrap the **img** element within a **figure** element and provide an optional caption using **figcaption**:

```
<figure>
  
  <figcaption>Caption for the image</figcaption>
</figure>
```

The caption is not the same as the **alt** text in the **img** element. The **alt** text is used to provide an alternative textual description of the image for assistive technologies, while the caption can be anything you wish.

Unstyled, the **figcaption** text is printed out beneath the image in plain text. You can use CSS to style it in italics, smaller print, or in whatever way you wish. The software I'm using for writing this book uses **figure** and **figcaption** as ways of attaching the captions to the images in all of the figures.

Adding Links to Video and Audio Files

HTML5 brought us two game-changing new elements: the **audio** and **video** elements. Not only do the new elements free us from having to use proprietary (and frequently problematical) technology, they're also quite simple to use.

The **audio** and **video** elements are similar. Both take a **src** attribute for the primary (or possibly only video or audio file). Both also have a **controls** attribute, which determines whether a control panel is displayed or not. Both elements also support a child **source** element, if you're providing videos or audio files in multiple formats. Not all browsers support all formats for both type of multimedia, and the way to ensure the farthest reach is providing multiple file formats.

An example of an **audio** element is:

```
<audio src="soundfile.mp3" controls preload="auto"></audio>
```

The source file is an MP3 file, which should work in all the major browsers, except Opera. The control panel is displayed. The **controls** attribute doesn't need to have a value assigned to—it's very presence is all that's necessary. The **preload** attribute is set to **auto** to

buffer the file before playing. Other values for this attribute are `none` (for no buffering) and `metadata` (for pre-loading metadata only).

Other audio file formats supported are WAV and OGG. If you have a source file in one format, such as MP3, there are conversion tools to transform it into the other formats. If you want the audio file to work in all of the browsers, then you'll need to use the **source** element and should provide (at minimum) the MP3 and the OGG formats:

```
<audio controls>
  <source src="audiofile.mp3" type="audio/mpeg" />
  <source src="audiofile.ogg" type="audio/ogg" />
  Your browser doesn't support the audio element
</audio>
```

The **source** element gets the source file name in the **src** attribute, and the type of audio file in the **type** attribute. Providing both MP3 and OGG files ensures the audio file is now playable in all of the major browsers. For the rare instance when the browser/client doesn't support either MP3 or OGG, the text contained in the **audio** element is displayed, as backup.

There are several audio element attributes, but the only other ones I want to cover in this introduction are the **loop**, **muted**, and **autoplay** attributes. Use the following setting if you want an audio file to play as soon as the page loads and the sound to continuously loop:

```
<audio controls loop autoplay>
  <source src="audiofile.mp3" type="audio/mpeg" />
  <source src="audiofile.ogg" type="audio/ogg" />
  Your browser doesn't support the audio element
</audio>
```

I'd go easy on this type of setting, though. Many people access web pages in an environment where sound blaring out suddenly, and continuously, could be a problem. Of course you could add in the **muted** attribute, which means the sound file will automatically play and loop, but you can't hear it until you turn on the sound.

Each browser company provides a different control display. **Figure 4-3** shows a montage of browsers (Opera, Chrome, IE, and Firefox) and their default audio controls.

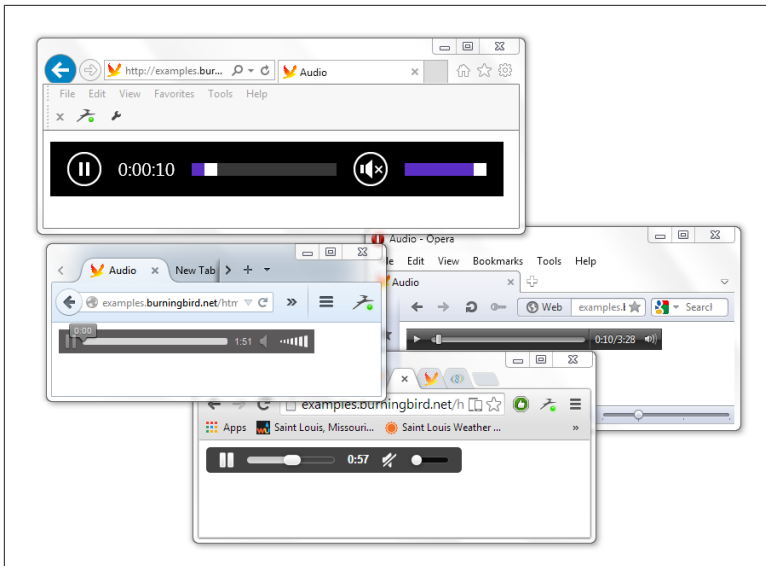


Figure 4-3. Default audio control in IE, Opera, Chrome, and Firefox on Windows 7

The **video** element is very similar to the **audio**. Again, you can specify a default video file, or use the **source** element to provide alternatives. Use the **controls** attribute to display a control panel, **preload** to buffer the file, **autoplay** to automatically play on loading, **loop** to continuously loop, and **muted** to start it silently. Three other attributes of interest at this time are **poster**, **width**, and **height**. Since a video has a display, you can control the element's width and height in the page. You can also provide a static image to display before the video is played:

```
<video controls width="640" height="480"
poster="birdcageposter.jpg">
  <source src="birdcage.mp4" type="video/mp4" />
  <source src="birdcage.webm" type="video/webm" />
</video>
```

You can set the source of the video using the **src** attribute, but just as with the **audio** element, not all browsers (and browser versions) support the same video type. You'll typically use the **source** element.

Most browsers (and Opera, with a later version) support the MP4 video type. Chrome, Opera, and Firefox also support the OGG and

WebM video types. In the example, both the MP4 and the WebM versions of the video are provided, covering all our bases. There are software and websites that provide free video type conversion if you only have video in one format.

Figure 4-4 shows a web page with a video element loaded into Internet Explorer. Note both the poster image that displays when the video is first loaded, as well as the controls.

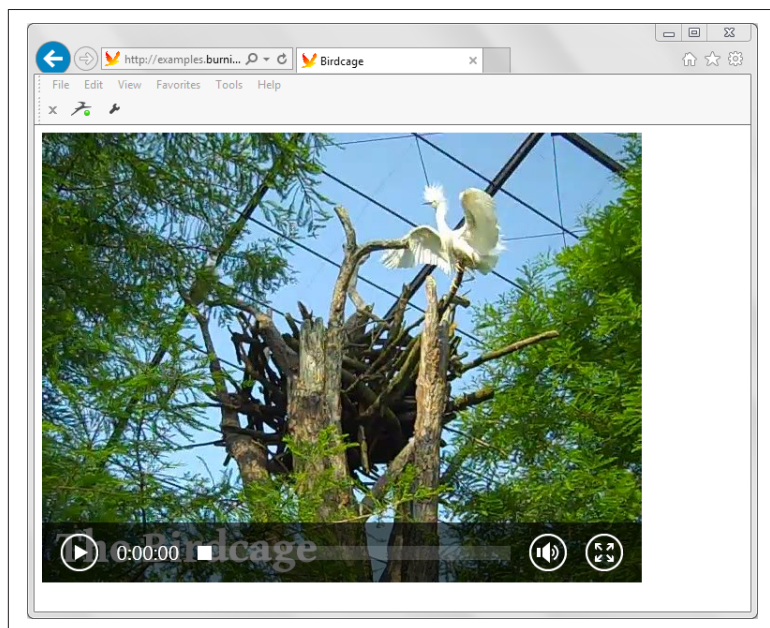


Figure 4-4. Video element loaded into IE

The media elements support one other child element I wanted to briefly touch on: the **track** element. It can be used to provide foreign language versions of a media file, as well as subtitles and captioning for accessibility. The following **video** element uses several **track** elements to provide both foreign language subtitles and captioning:

```
<video controls width="640" height="480"
poster="birdcageposter.jpg">
  <source src="birdcage.mp4" type="video/mp4" />
  <source src="birdcage.webm" type="video/webm" />
  <track kind="subtitles" src="birdcage.fr.vtt"
        srclang="fr" lang="fr" label="Français" />
  <track kind="captions" src="birdcage.en.hoh.vtt" srclang="en"
```

`</video>` `label="English Captioning">`



More on Video and Audio

The Mozilla Developer Network provides a good [overview of audio and video](#), including more detail on the media types. Sitepoint has an [article exploring the track element](#), including its benefits for search engine optimization.

Using a Static Page Template

In [Chapter 2](#) we created a fairly simple web page. It's readable, and not too nasty. But it's not really a web page you're going to want to keep around forever.

You can take a shot at creating your own design, but it's easier to start with an existing design and then modify it to suit your taste. Thankfully, there are several locations for freely available design templates. When you find one you like, you'll download a ZIP file with HTML, CSS, and graphics, adding your own content into the HTML file(s).

As an example, one site that provides hundreds of thousands of free static page templates is the [Open Design Community](#). You can browse through the designs or search for a design based on purpose, keyword, color, etc.

One of the more popular designs is [HTML5 Streets](#). It's a good design to use when learning how to use a template, as the HTML and CSS are clean and easy to read.

Once the template's zipped file is downloaded and the contents are extracted, open up the template folder. You'll find subdirectories labeled *js*, *fonts*, *images*, and *css*. In the *css* subdirectory is a file, *style.css*, containing the CSS for the design. When you get a chance, take a look at the style settings, because an excellent way to learn CSS or HTML is from example.

In the main directory are the template's HTML files:

- *contact.html*
- *index.html*
- *ourwork.html*

- *projects.html*
- *testimonials.html*

They're just a beginning set of HTML files. Links to each are included in the main *index.html*. You can change the files, the link text, or remove one or more of the files and links. If you open the *index.html* file in your favorite text editor, you'll see the web page structure with content that you can replace with your own text.

Once you're finished with changing the text, upload all the template files and subdirectories to your site, and see how it looks. **Figure 4-5** shows the template uploaded to the *blipdebit.com* website, opened in Firefox.

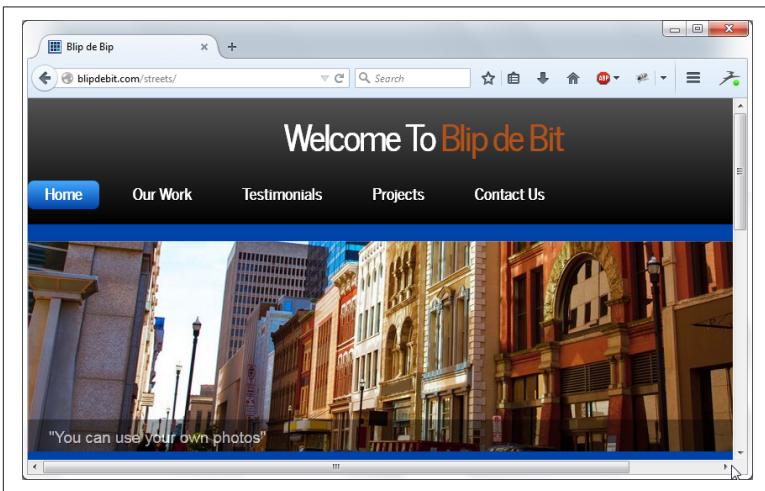


Figure 4-5. Using a static template to create a website

There are templates for any purpose and many different types of businesses. They're a good way to jump-start your static website, eventually modifying it until it is uniquely your own.



Static Versus Dynamic Websites

I keep mentioning static websites, templates, and pages. Static web pages are ones hand edited with information that only changes when you manually change it. In **Chapter 5**, I'll demonstrate how you can use software to create dynamic websites and pages.

Adding a Simple Mail-To Form

Most websites, even the simplest, provide a way for your people to contact you. You can embed your email address in the web page, but that's opening you up for a significant increase in spam (junk) email.

A better approach is a contact or *mail-to* form. Thankfully, most shared hosting companies provide a simple one button solution for building a mail-to form. BlueHost calls their mail-to form Bluemail, and it starts in cPanel.

In the cPanel interface, you'll find a set of options under the label Software/Services. Within these options is one labeled CGI Center. CGI stands for Common Gateway Interface; it is the oldest form of web technology and still in wide use. Clicking it opens a page that describes the BlueMail CGI application.

As the page notes when you open it, you must have at least some familiarity with web forms, which is what I'm going to provide in this section. BlueHost provides the backend processing of the mail-to form, but you'll need to add the HTML form to your own web page. BlueHost provides a sample form already created, which we can copy and paste into our web page and modify as needed.

The sample code provided is:

```
<form action="http://www.bluehost.com/bluemail"
      enctype="multipart/form-data" method="POST">
  Name: <input type="text" name="Name"><br>
  Email: <input type="text" name="mailfrom"><br>
  Street Address: <input type="text" name="StreetAddress"><br>
  City: <input type="text" name="City"><br>
  Zip: <input type="text" name="Zip"><br>
  Phone: <input type="text" name="Phone"><br>
  <input type="hidden" name="sendtoemail"
    value="webmaster@blipdebit.com"><br>
  <input type="submit" value="Send Email">
</form>
```

Let's explore each of the elements in the sample form, in turn.

To start, all of the form component elements are enclosed in the outer **form** element. It has three attributes, which we need to leave as is for the BlueMail mail-to form to work. The first is the **action** attribute. This is the attribute that instructs the browser what to do

when the web form is submitted. In this case, the form data is sent to the pre-built BlueMail CGI application. The second attribute is **enctype**. It specifies the encoding scheme used for the form, in this instance, `multipart/form-data`. This is the most commonly used encoding scheme, and is essential if one of the form elements is a file to be uploaded. The last attribute is the form **method**. It tells the browser whether the form is sent using a POST method or GET. Since you're uploading data to the server, you'll typically use POST. The GET method is used primarily when you're retrieving data from the server to display in the page.

What follows next is a series of text labels and input elements. The input elements are where the web page form user types in the values sent in the mail-to form. In the form, the input types are all text, but input elements can have the types described in [Table 4-1](#).

Table 4-1. HTML input element types

Type	Description	Visual control
hidden	Provides a means to send data to server without user input	None
text	Sending arbitrary text (no line breaks)	Text field
search	Sending search text (no line breaks)	Search field
tel	A telephone number	Text field
url	A URL	Text field
email	An email	Text field
password	Arbitrary text with no line breaks	Input is obscured
date	A date (day, month, year)	Date control
time	A time (hour, minute, seconds, fraction of seconds)	Time control
number	A number	Text field or number spinner
range	A numerical value	Some form of slider
color	An RGB (red-green-blue) color	Color picker
checkbox	Zero or more options	Check boxes
radio	Zero or one option	Radio buttons
file	For uploading a file	Button and label
submit	To submit the form	Button
image	An image	Clickable image or button
reset	To reset the form fields	Button

Type	Description	Visual control
button	Do some action	Button

Some of the input types, such as **color**, **date**, and **time**, are new with HTML5 and aren't supported in all of the major browsers. Others, such as **email** and **tel**, will not only accept text formatted as an email or telephone number but also issue an error message to users if they type in incorrect text.

The BlueHost mail-to form uses **text**, **hidden**, and **submit** input types. The **text** fields are for the user to type in contact information, the **hidden** field contains the email that receives the contact information, and the **submit** button submits the form. You can modify the email address used in the **hidden** field, and you can add the following field to redirect to a specific web page when the form has been processed:

```
<input type="hidden" name="redirect"
value="http://www.**redirect url**">
```

Change the URL to the one you want.

Copy the entire HTML form and fields and paste it into the contact web page or wherever you want to place this mail-to form. Now anyone who wants to contact you can, without you having to embed your email into the web page. It also provides a clean point of contact.

You can modify the form and use CSS to style its appearance. Most hosting companies provide one or more CGI applications you can incorporate into your website without once having to touch the code. Just look for the CGI icon in cPanel.

Adding Dynamic Content

Static web pages (pages that are manually updated) can be good for many uses, but if you want to generate activity about your website, you're going to need to provide new, useful, and frequently updated content. To do this, you need software that can dynamically generate the content. Typically, you'll need weblogging or other content management system (CMS) software.

Once your site is active, then it's time to make sure people can find your site, either via a search engine or using social media.

Creating a Subdomain For Your Weblog (or Other Purpose)

Weblogging software got its start over 15 years ago and is now ubiquitous. It ranges from more formal (and complex) CMS tools, such as Drupal, to the now very popular Wordpress. In this chapter, you'll use your hosting company's cPanel to install Wordpress.

Before we start, though, we're going to create a *subdomain* for your new weblog. A subdomain is a domain name such as *weblog.youdomain.com*. It's a way of using software, such as Wordpress, without having to overwrite any existing static front pages. We installed static web pages using a downloaded template in [Chapter 4](#), and we don't want to lose that effort.

To start, log into your hosting account (Bluehost for the example), and access your cPanel. In it, you'll find a group of icons labeled

Domain Management. One option is for adding and managing subdomains. Clicking this opens up a form for creating your new weblogging subdomain.

For *blipdebit.com*, I decided not to use the more traditional *weblog.blipdebit.com*. I picked *stuff.blipdebit.com*, instead, because it sounds more fun. You can choose whatever works for you.

When you type in the subdomain name, the tool automatically generates a subdirectory for the subdomain, with the same name as the subdomain. Every subdomain has its own subdirectory. Unless you have a reason not to use the generated name (such as a subdirectory of that name already in use), accept the folder name, as shown in **Figure 5-1**.

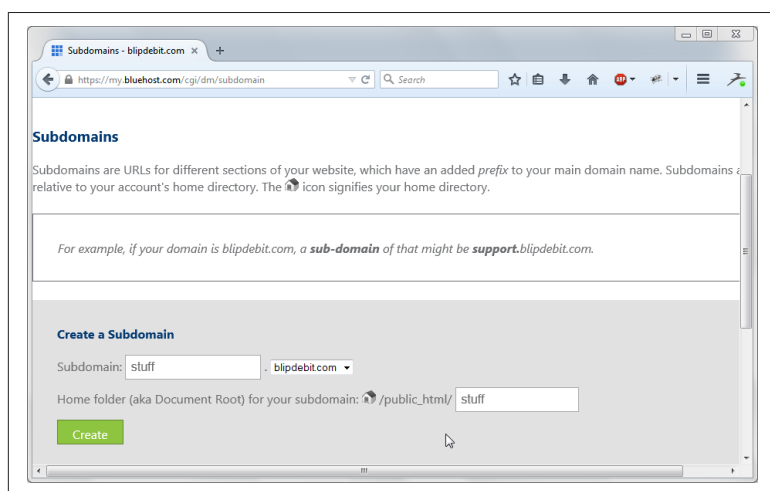


Figure 5-1. Creating a subdomain for a weblog

It takes a few minutes for the subdomain to be accessible. When it is, you're ready to create your weblog.

Creating the Wordpress Weblog

Returning to the cPanel, look for the group of icons labeled Website Builders. These are all the software available to you for creating website content. Some of the builders, such as Weebly, allow you to dynamically drag and drop sections for creating new web page templates; others, like Wordpress, provide *themes* for controlling your site's look and feel.

Click the Wordpress option. In the page that opens, click the Install button. In the next page, you'll be asked for the domain where Wordpress is installed. If you click it, it opens up a selection box listing all possible domains, including your newly created subdomain. Select that option.

In the next page that opens, you'll be given an option to choose advanced features. Check that box. In the area that opens, you can add your weblog name and provide an admin username and password, as shown in [Figure 5-2](#). Provide your own website name, admin username, and password. Also check the box labeled "Automatically create a new database for this installation." You'll want the tool to create a database for you. Also check the option labeled "I have read the terms and conditions of the GPLv2." This is the licensing information for using the software.

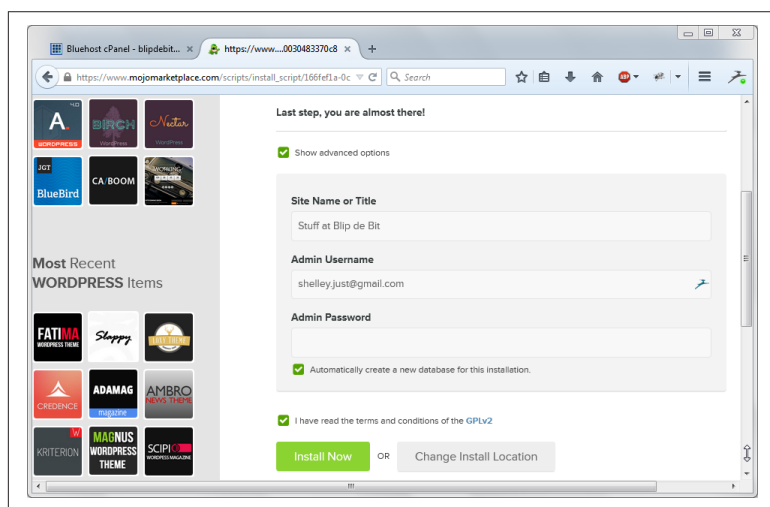


Figure 5-2. This is an image caption

The next page that opens lets you know the installation is under way and presents you a set of paid themes. There are thousands of free Wordpress themes, so don't feel you have to pay for a theme. And you're going to want to get familiar with your weblog before you install any extra plugins or themes, so ignore the page for now.

Accessing your weblog after Wordpress is installed, you'll see an automatically generated web page stating that the new weblog is coming soon. Administrative pages can then be accessed via the *wp-admin* subdirectory, as in *stuff.blipdebit.com/wp-admin*.

Beware the Paid Stuff

If it seems like you're being hit in the face with stuff to buy every time you turn around in your hosting company's pages, you are. And most, if not all, of the hosting companies insert opportunities for you to spend money in almost every circumstance. It can get frustrating and, sometimes, even a little irritating.

Don't automatically reject the options, but also don't feel you have to go the professional or paid route. For instance, you don't have to use a paid theme with Wordpress. However, if the theme you fall in love with is a paid theme, it's a good use of money. And if you're uncomfortable putting your Wordpress site together, definitely check out the options to hire folks to do the work for you. They'll charge you a flat fee to set up your weblog or other site offering, with the options you want, and you'll be ready to go.

But if you're happy to take a swing at things yourself (most of the software is quite simple to use), ignore the blandishments to spend money, and have fun.

Exploring Your New Weblog

The Wordpress administration page is similar to that shown in **Figure 5-3**. The page may contain a banner asking you to connect your site to Wordpress.com in order to benefit from statistics, social media, etc. You can do this later, so go ahead and close the banner, and let's explore the page's offerings.

Along the top are options to go directly to the weblog's home page and to view the comments awaiting moderation. There's also a plus sign (+) that provides a quick way to add a new post, page, or user, or to upload a media file. Next to that is customization associated with the vendor who provided the Wordpress installation. You can ignore that for now.

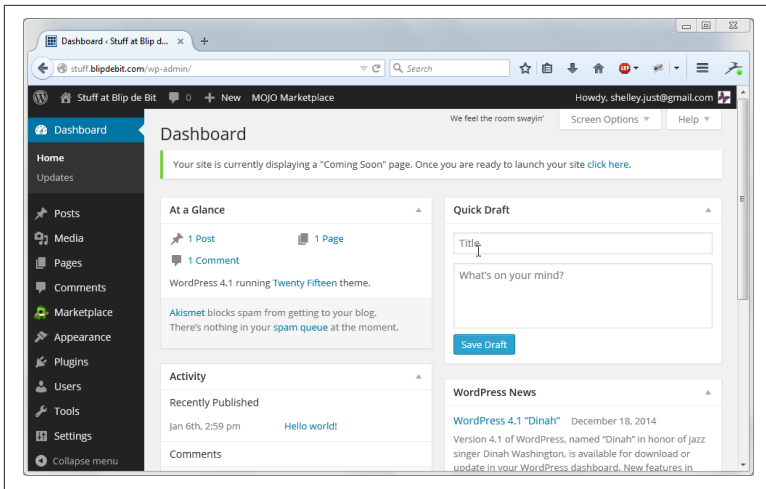


Figure 5-3. The Wordpress weblog dashboard

On the left side are options to access the Dashboard, where you can see the state of your weblog at a glance, and Updates, which provides notices of a new installation of Wordpress or whatever plugins you're using. The really nice thing about Wordpress is all software updates are accomplished with just a few clicks of the button.

Scroll down to the bottom of the left sidebar, until you see the Settings option. Clicking this opens up the settings for the weblog, displayed in Figure 5-4. In it you'll see the weblog name, a place to provide a tagline to go with the name, and to include your email address for any weblogging emails. There are also fields to set the site's date and timezone and whether you allow new users. You don't have to allow new users in order to allow commenting, so I'd leave that off for now. Do adjust the other parameters to match your location.

The Appearance option allows you to define new menus for your weblog, change the header, widgets, theme, and do an overall customization of the weblog's appearance. We'll cover these next.

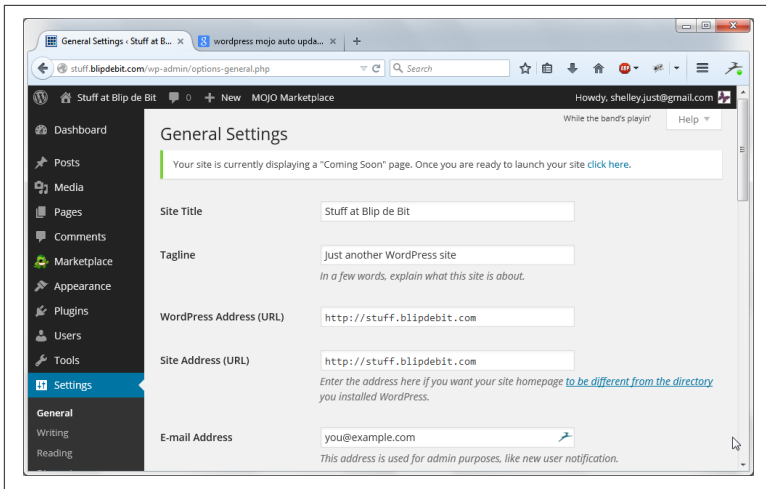


Figure 5-4. General settings page for the new weblog

Choosing Your Weblog's New Look

You have significant control over the appearance of your weblog. To start, you can use any number of pre-built Wordpress themes and then add your own customization.

The Themes page opens when you click the Appearance option in the left sidebar. The page displays the theme you're using: the default Twenty Fifteen, which is a nice, uncomplicated theme. Also installed are other default themes from which you can choose. To change the theme, just click on the one you want, and click the Activate button that opens, as shown in [Figure 5-5](#).

If you don't like any of the pre-installed themes, from within your Wordpress installation, navigate to the Wordpress.org website, and you can browse among the many available themes. At the site, you can specify the features you're interested in, such as *responsive design* or accessibility, and those themes that match are displayed. If you find one you want to use, click the choice and then the Install button that displays. Clicking that button installs the theme into your site. Once it's installed, it will appear in your list of available themes.

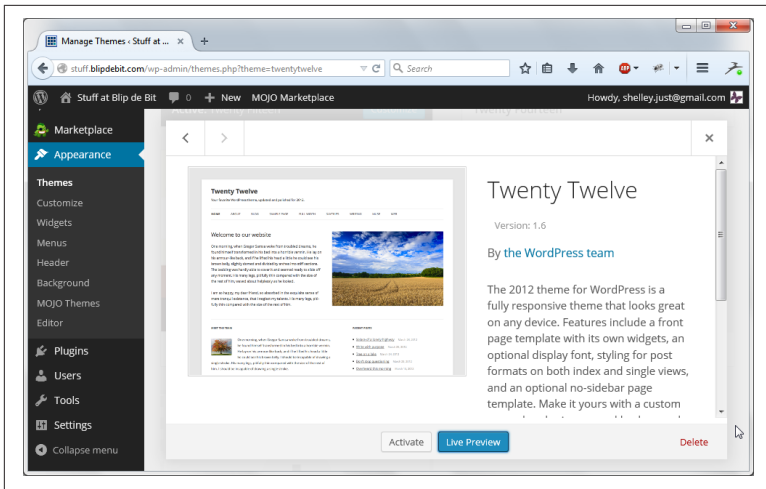


Figure 5-5. Activating a different theme

After you've picked a theme, you can customize it. For example, you can customize the Twenty Thirteen theme by clicking the Customize button associated with the theme. In the page that opens, in **Figure 5-6**, the customization options are in the left sidebar. You can modify header color, the header image, widgets, and other components of the page.

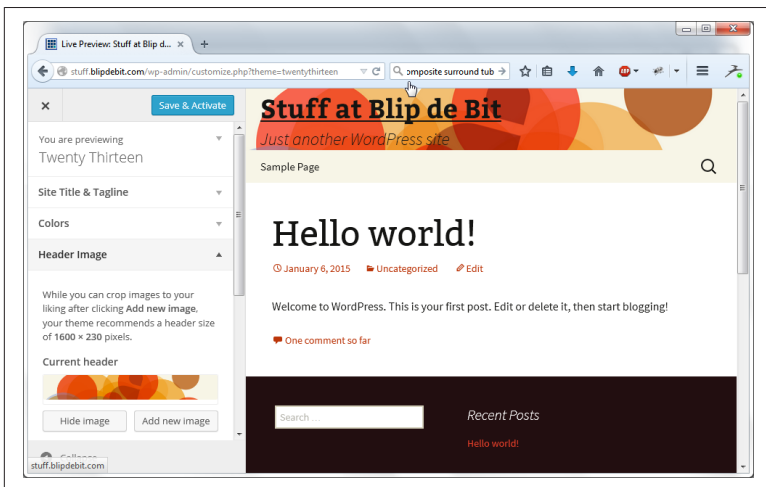


Figure 5-6. Customized Twenty Thirteen theme used for weblog

For the examples in the rest of the chapter, I use the Twenty Thirteen theme as is, except for not using a tag line. Next up: adding some toys.

Responsive Design and Media Queries for Mobile Designs

A website is just as likely to be viewed using a mobile device as a computer. Your website has to adapt to meet the needs of devices with ranging screen sizes and computing resources.

Luckily, many themes and static design templates are created using the concepts of *responsive design*—design techniques geared to ensuring the web page looks good regardless of the screen size of the device. The approach utilizes flexible design constraints, such as widths based in page size percentages rather than inflexible fixed widths.

Media queries can assist in the design. A media query allows us to define criteria for stylesheets to be loaded or page elements to display or other CSS styling, based on screen sizes and other criteria. An excellent example from [Mozilla's Developer Network](#) is the following:

```
<!-- CSS media query on a link element -->
<link rel="stylesheet" media="(max-width: 800px)"
      href="example.css" />

<!-- CSS media query within a stylesheet -->
<style>
  @media (max-width: 600px) {
    .facet_sidebar {
      display: none;
    }
  }
</style>
```

In the code, the linked stylesheet is only applied if the maximum width of the browser window is 800 pixels. The stylesheet still loads, but it isn't applied. In addition, the sidebar is hidden if the maximum window width is 600 pixels or less.

If mobile support is important, check to see if a theme identifies itself as being a responsive web design or otherwise mobile-friendly, when you're shopping for a new theme.

Adding the Perfect Site Widgets

Widgets are self-contained useful little bits of functionality built into Wordpress. Clicking on the Wordpress Widgets option displays widgets currently installed and their location in the web page. For the Twenty Thirteen theme, the primary widget area is the footer, while the sidebar is the secondary widget area. Each theme has different widget areas.

The theme has the following widgets installed:

- Search
- Recent Posts
- Recent Comments
- Archives
- Categories
- Meta

Most of the widgets are self-explanatory. The Meta widget provides a link to the Site Admin page, as well as links to RSS, for those who are interested in subscribing to your feeds.

To remove an active widget, click it, and click “delete.” To add a new widget, click the available widgets, and then activate it in the widget area you prefer. You can also use the Customizer, which simplifies widget addition and removal. The Customizer for the weblog is shown in [Figure 5-7](#). Currently there are no widgets in the sidebar, so it isn’t displayed. The theme is a modern, minimal, responsive design, and this type of design doesn’t favor sidebars.

I like both the footer-based widgets and the current selection, so I’m leaving the widgets as is. If I can’t find the functionality I wanted among the installed widgets, I would then need to look for it among the plugins, covered next.

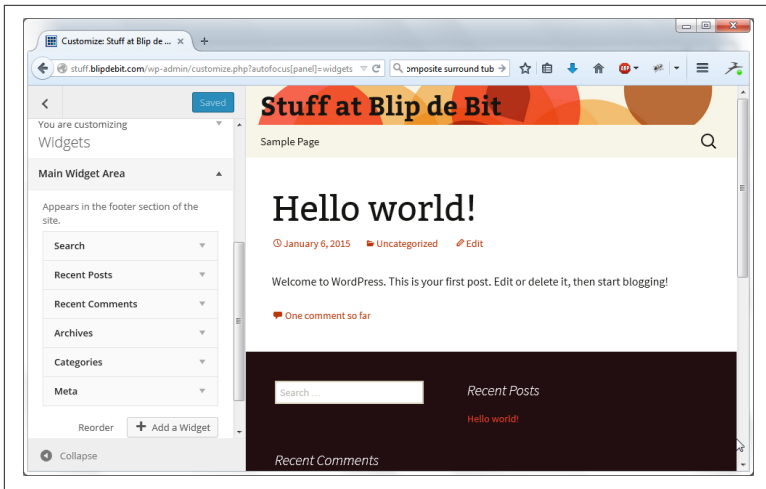


Figure 5-7. Checking out installed widgets

Adding a Plugin

Additional functionality can be added to your weblog via a *plugin*. Clicking the Plugins link in the Dashboard displays the plugins installed, and provides a link to add new ones. You'll also get a banner notice about activating your Akismet account. Akismet is an anti-spam service, which I'll cover later when we discuss adding weblogging comments.

If you installed Wordpress using MOJO, plugins from the MOJO Marketplace are listed. Here is where you can disable the MOJO additions that appear everywhere. There's also an option to enable the connection to Wordpress.com. Last, but not least, the famous Hello Dolly plugin has been around since Wordpress's early days.

To find new plugins, click the Add New link at the top of the page. As happened when we were browsing among the themes, you can browse among the many plugins available. You can also use the page that opens when you click the link to upload Wordpress-compatible plugins that you've downloaded as *.zip* files.

Among the plugins available at Wordpress are a couple that stand out. One is Wordfence, which promises to make the site more secure and much faster. And it's free, so we'll add that one. There's another one, Editorial Assistant, that finds related posts and images

as we write. That one also appealed, so I grabbed it. There are others, but it's best to start small, and add additional plugins later.

Returning to the primary plugins page, we can see the two new plugins we installed. Now, all we have to do is activate them, as shown in **Figure 5-8**. That, and explore what they do. Some plugins don't require further explanation in how they work (Editorial Assistant), while others do (Wordfence).

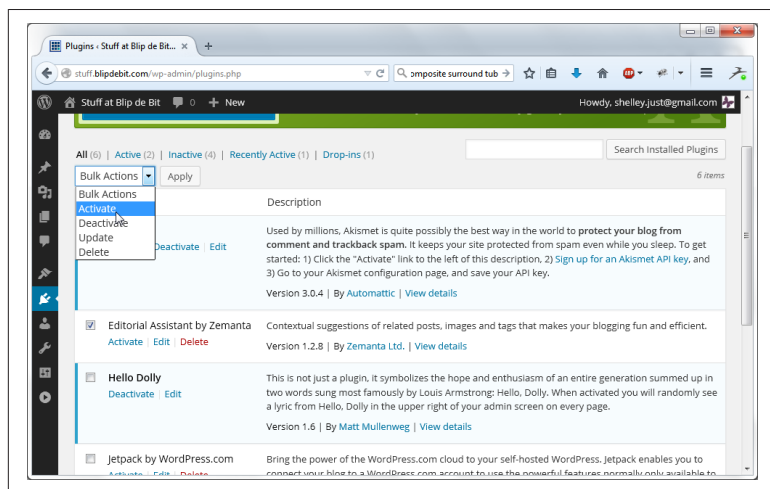


Figure 5-8. The Plugins page after plugins have been installed

Some of the plugins, such as Wordfence, will also add a left sidebar menu icon. The icon can appear anywhere, but the Wordfence icon appears after the Settings icon in the WordPress Dashboard.



Free as in Freemium

It's worth noting that Wordfence is a *freemium* plugin, which means you'll get emails encouraging you to upgrade.

Now that we've added a new theme, and a couple of new plugins, let's add some content.

Creating Wordpress Posts and Pages

There's a quick option button labeled with a plus sign (+) in the top of the Wordpress Dashboard page to add a new post, media file,

user, or page. There's also links on the side to open the administrative pages for each type of resource. Click the icon for Posts (a thumbtack).

Posts

The most recent posts are shown in a table, along with a button next to the page title to add a new post. A dummy post was created when the weblog was created, and clicking it opens the page in the editor, as shown in [Figure 5-9](#).

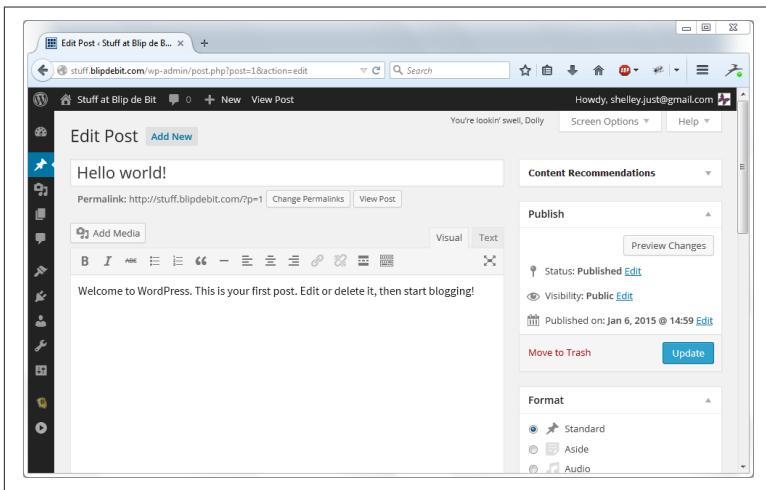


Figure 5-9. The Edit Post page

The left side shows the post title and the content. Wordpress provides a visual editor that is capable of accepting text and provides buttons for special annotation. To use the buttons, select text, and hit the button. You can make the text bolder or italic or insert a list, link, or horizontal bar. You can also insert a Read More tag. This is a handy way of setting what text will show on the front page, with a Read More link for people to access the full post.

Clicking the Toggle Toolbar opens up a second toolbar with even more options, including adding special characters into the page. If you click on the Text tab header, you'll then see the actual HTML for the content. There's an X-like expander character in the rightmost position in the top toolbar. When you click it, all the material surrounding the post is hidden so you're not distracted.

The right side controls the metadata for the post. Each block can be collapsed, so you're only seeing what you want to see. The first block is a Content Recommendations block, which you can explore later. The second reflects the post status. You can switch between published and not, whether it's publicly visible or not, and the date it's published. All the values can be edited.

Beneath the status block is a block containing several formats. These just provide additional metadata for the post, which themes can use to provide specialized formatting. So if your post contains a video, picking the video format can trigger specialized design features.

The next block is for selecting a category or categories for the post. You can also use the block to create a new category. The same for the next block for creating a tags. You can use an existing tag or create a new one.

The very last block is used to set an image for the post. Nowadays, associating an image with a post is very popular, and this feature makes doing so a snap. All you have to do is upload the image, provide a caption, and the tool inserts it into the page for you.

When you're ready to save the page, you can modify the *permalink* that Wordpress automatically provides. A permalink is nothing more than the URL used to access the specific post or page. You can use a more meaningful URL for the permalink, such as *http://stuff.blipdebit.com/your-article-title*, rather than the default of *http://stuff.blipdebit.com/?page_id=8*.

Posts are published in reverse chronological order on the front page of your weblog. Ten posts are displayed, by default, on the front page at a time, but you can alter this by going to Settings, Reading, and changing the value in the page. You can also alter whether syndication feeds get the full text or only the summary. If you wish, you can check whether to discourage search engines from indexing this page. If your weblog is a private space for close friends and family, check this option.

Posts are writings equivalent to articles, status updates, or diary entries. They are the dynamic component of a weblog. For more static content, such as an About page, you'll want to create a page.

Pages

Open the Pages administration page by clicking the Pages icon in the left sidebar. The icon is a stack of pages. The administration page is similar to that for Posts: a table with the most recent pages, and a way to add a new one. Since most weblogs have an About page, click the Add New button to add one.

The editor for the page is the same as the editor for the post, as shown in [Figure 5-10](#). It has the same toolbars and ability to change the permalink, view the actual HTML, and to toggle toolbars. The main difference is in the blocks available on the right side. There are no categories, tags, or formats. You can set the page image, publish the page, or set page attributes. The page attributes are a way of nesting pages, by specifying a parent and giving the order. For the most part, pages aren't nested, and their order is irrelevant.

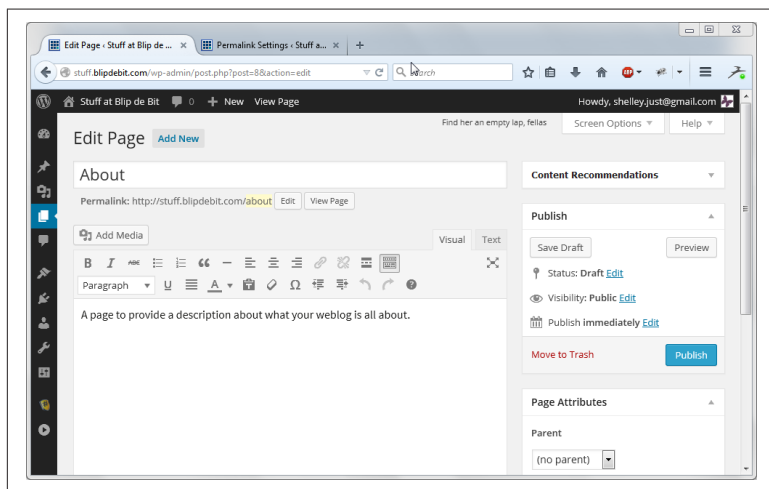


Figure 5-10. Creating a new Wordpress page

When I created a new About page for *blipdebit.com*, I provided a brief blurb describing the purpose of the site. I also changed the permalink for the site to <http://stuff.blipdebit.com/about>, since this is more meaningful and generally easier to type.

Speaking of which, rather than have to change the permalink for each post or page, I can change them universally. Clicking the Settings icon in the left side of the dashboard and then choosing the Permalink option opens the Permalink Settings page. I can choose

from a number of options, as shown in **Figure 5-11**. I can choose to display the day and name, month and name, post name (where spaces are replaced with dashes), or provide a custom format using any number of permalink tags. You can also define a default permalink for categories and tags, as these can be accessed directly, too.

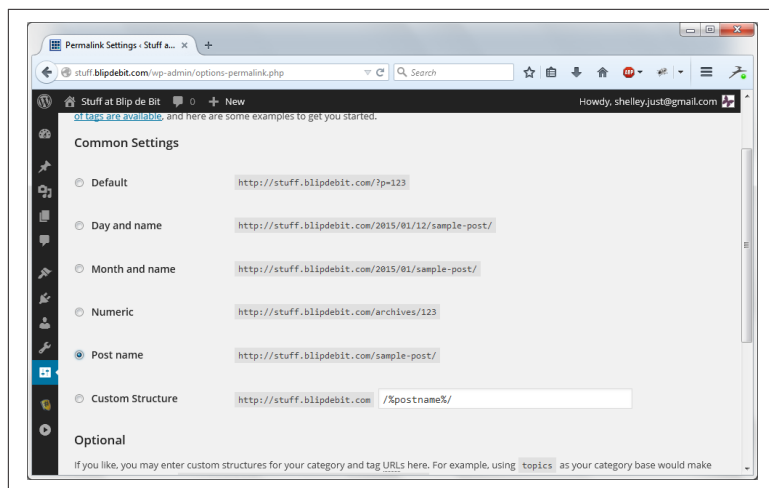


Figure 5-11. Providing a default permalink structure for posts

Incorporating Social Media Into Your Site

Though you're no longer depending only on social media to put your message online, social media can still be an effective ally.

You can integrate your website and the popular social media sites, such as Facebook and Twitter. If you have a personal Facebook page, update your profile to point to your new website. If you don't have a Twitter account, you can get one and tie it to your website.

You can go beyond these simple measures, with a little help from plugins. Some of the plugins allow you to publish your weblog posts to your Facebook timeline, but I have to warn you that the hoops you have to jump through to accomplish this are excessive. And seemingly, ever changing.

A better way to ensure that your material is posted on social media sites is to add sharing buttons for each site and to provide a set of icons where people can find you on each site ("follow" buttons). You can find and install plugins for each individual social media site, or you can install a plugin that manages most or all for you.

Automattic's Jetpack Plugin

Automattic, the creator of Wordpress, provides a many-purpose plugin named Jetpack, providing all sorts of useful functionality.

If you want to alert various social media sites about a new post, you can use the Jetpack Publicize option and notify them all at once. You can also add more widgets, check out your site's statistics, create customized photo galleries, simplify sign-on, add a mobile theme, and so on.

Many people consider the plugin to be indispensable, so it should be one of the first you check out.

I decided to go for one plugin that does it all: the Ultimate Social Icons and Share Plugin. I liked it because of all its options and because it's one of the few that isn't trying to sell you something. To install it, click the Add New link in the Plugins page, and search "Ultimate Social Icons and Share." Click the Install button with the plugin, and activate it once it's installed.

The Ultimate plugins places a link in the left sidebar of Dashboard. When you click it, the page that opens, shown in [Figure 5-12](#), provides options to control both the share buttons in the post and the follow buttons, added to the web page's sidebar.

Expand each group to make choices and provide information. You pick which social media buttons you want to display, provide information (such as your Twitter username, Google+ profile page, or Facebook home page), select your design options, and you're good to go. If you want to display share buttons at the bottom of posts, click "Do you want to display icons at the end of every post?" and check the Yes button. You can also change the text to display with the buttons or choose to delete all text.

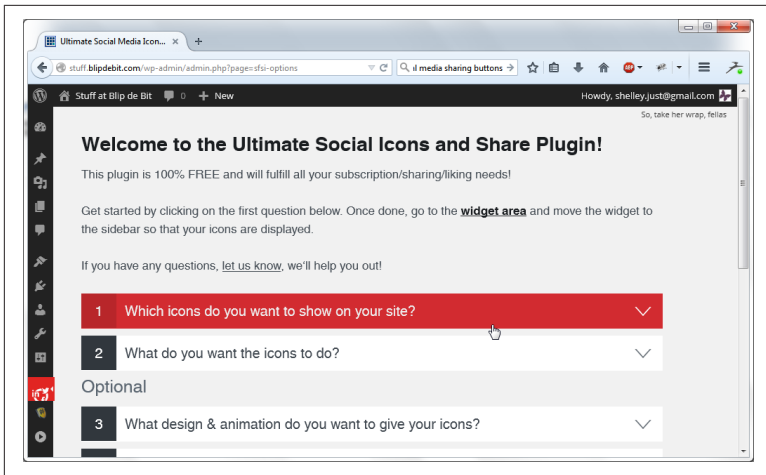


Figure 5-12. The Ultimate Social Icons and Share Plugin's settings page

Once you've defined the settings for the buttons, place the "follow" buttons using the Widgets option in the Appearance setting. In the theme I've selected, I could place them into the currently empty Secondary Widget Area, or in the Main Widget Area. I decided it was time to wake up the Secondary Widget Area, so I added the buttons and a few other widgets to it. I also deleted the text associated with the buttons, since I felt the buttons were self-explanatory. Figure 5-13 shows both types of buttons after all my tweaking.

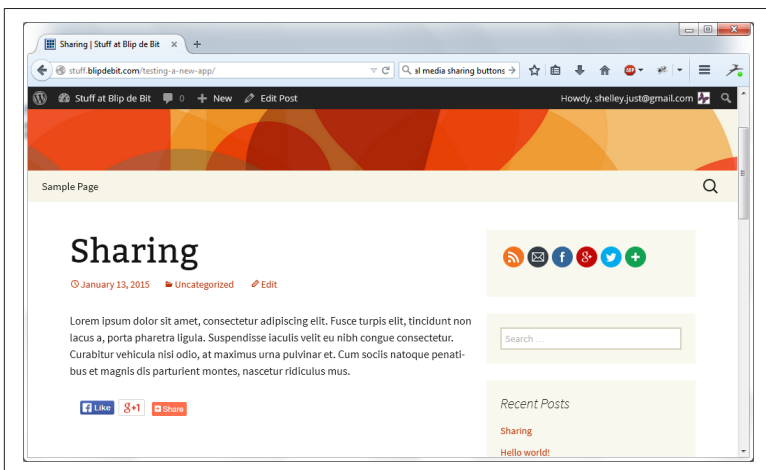


Figure 5-13. Newly implemented social media buttons

Helping People Find Your Site

Once your site is ready to go, you're connected to social media, and you're posting, you're ready to beat down that path to your door. The best way to introduce your writing to the world is to engage with people on social media, as well as post links to your work. It also helps to link to other people's work.

Make sure your titles reflect the topics you want search engines to zero in on. An effective use of categories and tags can also help. Good titles are also critical for social media sharing, since it's the title that people see in Facebook, Twitter, or Google+. Cute titles might be fun, but out of content, they're not informative.

There are paid-for services that promise to increase your ranking in search engines, but you'd be better off keeping your money. Get attention for your site by posting useful, interesting, entertaining, or targeted content. And the more you're engaged with others, the more they'll engage with your site.

From a technology perspective, it helps to provide an *XML sitemap* to make it easier for search engines to understand your site's infrastructure. If you think there's a Wordpress plugin for that, you're right. There are several you can check out, but I recommend you first check out the Google XML Sitemaps plugin (Figure 5-14).

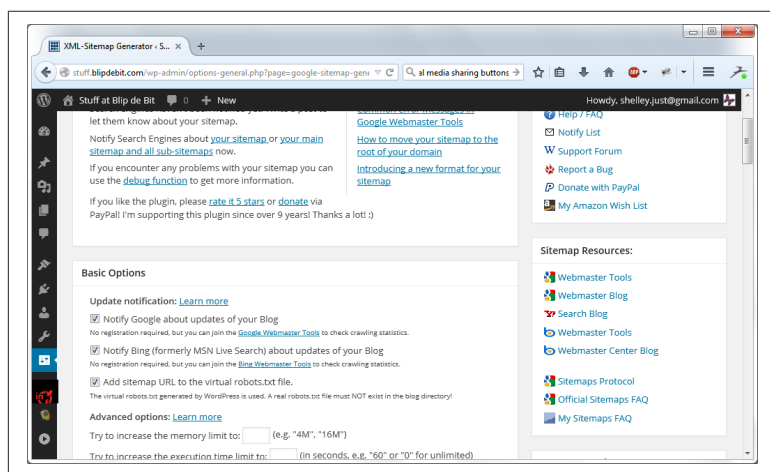


Figure 5-14. All the many options in Google XML Sitemaps plugin

Once activated, the Settings page for the plugin shows a plethora of options, including search engines to notify when you post, an option to create the Sitemap, links to Webmaster tools, and a host of other goodies. It is more than worth your time to explore in depth.

Adding Weblog Comments

The best way to generate attention in your site is to allow weblog comments. It can also be the easiest way to lose control of any message you want to communicate. Comments are, in nothing else, a double-edged joy.

Your weblog comes equipped with Wordpress comments. To enable or disable them, select the Discussion option from Settings. You can turn comments on, specify if only registered users can comment, whether comments can be threaded (replies are nested), and other settings, as shown in [Figure 5-15](#). I recommend that you automatically close comments after a week or so, and also that you moderate comments. One option is to moderate a comment, unless the individual posting it already has a previously posted comment. This could help eliminate spammers without having to moderate every comment.

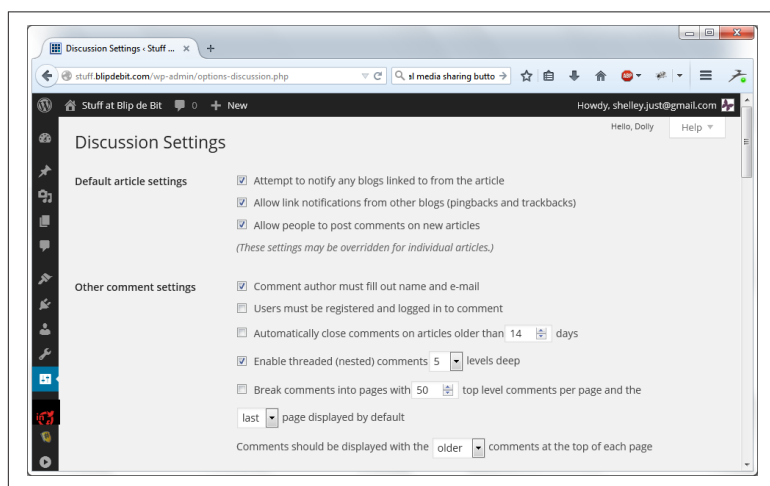


Figure 5-15. Wordpress comments

Speaking of spam, we can now finally cover that Akismet plugin that has been blasting us in the face every time we access the Plugins page. Akismet is useful for filtering spam out of your comments. To

use it, you have to get an Akismet key, which means you'll also have to sign up for a Wordpress.com account. Akismet is useful, but it is a subscription service. For a personal website, you can use Akismet for free or contribute a small sum of money to help the service. When you subscribe (for free or otherwise), Akismet provides a key to use with the Akismet plugin in your Wordpress weblog. You can choose how Akismet handles your spam, as shown in [Figure 5-16](#).

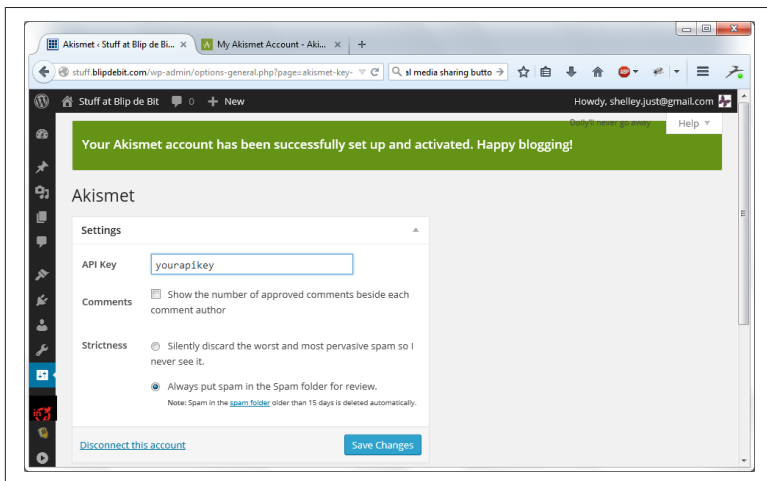


Figure 5-16. Telling Akismet what to do with your spam

You can use other commenting systems, such as Facebook's or Disqus, but give the Wordpress comments a try to start.

Advanced Web Technologies and Techniques

The first five chapters cover most if not all that you need to know for creating and maintaining a website, whether the site consists of static pages, a weblog, or both. There are times, though, when you need to go beyond the basics.

The File Manager

You've been able to upload files using FTP, but you can also work with the file system through cPanel applications available under the File Management label. You can check how much disk space you're using with the Disk Space Usage application, find out how many files are in each subdirectory with the File Count application, and manage the files themselves with the File Manager.

When you access the File Manager, it asks which directory to open, such as the home directory or the site's document directory. **Figure 6-1** shows the File Manager opened in the *blipdebit.com* home directory. One of its subdirectories is labeled *stuff*, and contains the files for the site's weblog.

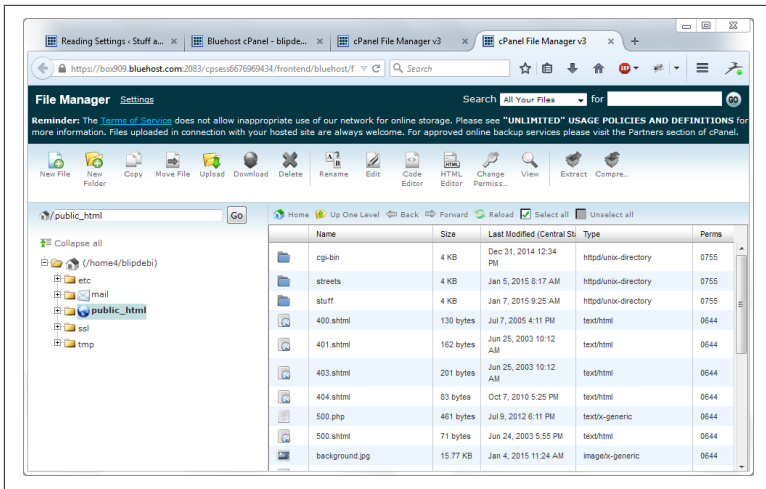


Figure 6-1. The blipdebit.com website home directory opened in the File Manager

Using the File Manager, you can create a directory or remove an existing one. You can also copy files, create new files, move them, or download files without having to use FTP software. If you want to edit an HTML file, click the file and select the HTML Editor. The file opens into a decent visual HTML editor, shown in **Figure 6-2**.

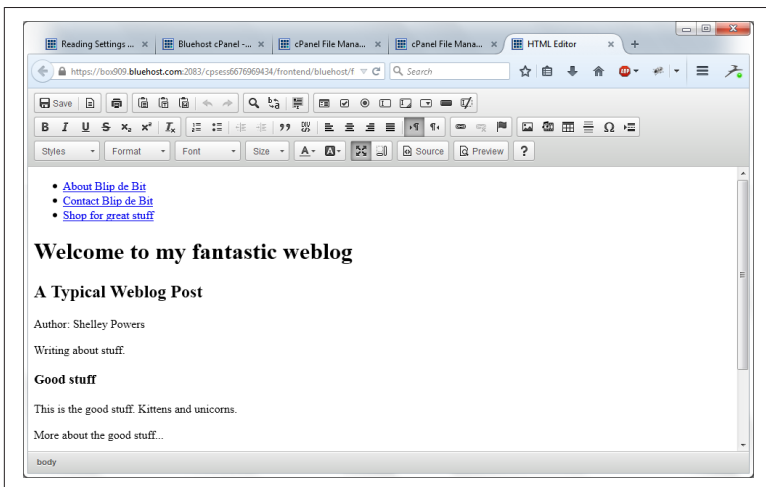


Figure 6-2. Web page opened in the HTML Editor

You can also use the File Manager to change permissions on a file. For the most part you shouldn't be concerned about file permissions, but some software may require you to change the permission to allow the software to write to the file or to remove write permissions for security purposes.

When you select a file and then access the File Permission option, a small window opens displaying all the files permissions across User, Group, and World, as shown in [Figure 6-3](#). The permissions are Read, Write, and Execute. When software asks to add or remove permission, you'll be changing the World, or the Group and World permissions. Just add or remove checkmarks according to what the software needs.

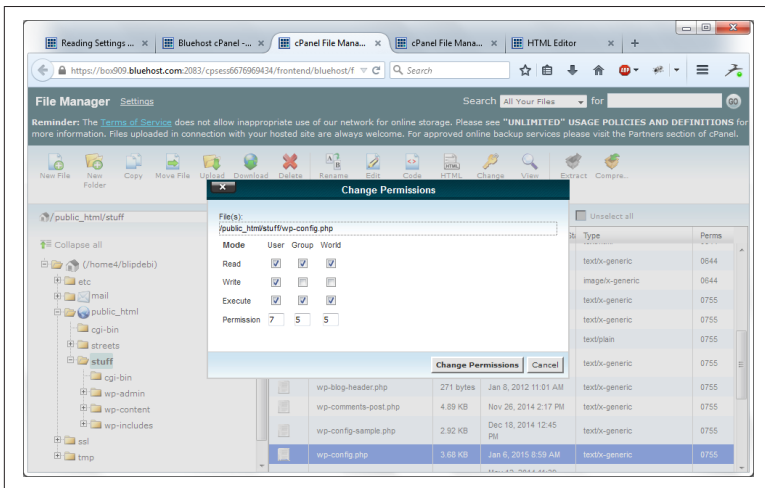


Figure 6-3. File permissions for the Wordpress *wp-config.php* file

Managing Recurring Tasks with Cron

Some software requires recurring activities. For instance, many content management systems (CMS) requires you to check if there are new updates on a regular basis. You can schedule the updates in your calendar and manually make them, but a better method is to schedule the task using an automated system: Cron. To perform the automated task, you create a *Cron job*.

To create a Cron job, in your cPanel, look for a set of tools under the label Advanced, and then click the Cron jobs icon. The writing at the top of the page notes that you do need to be familiar with Linux

commands, but for the most part, software that needs to have Cron jobs set up also provide instructions in what to input into Cron.



Linux Commands

If you're interested in learning about the basic Linux commands, I recommend [Linux For Newbies/Command Line](#). You can also play around with a [Linux command-line emulator](#).

To demonstrate how Cron works, create a text file with some text and load it up to your server. It doesn't matter what the text is, as it's only being used as an example. The command you're going to want Cron to run is the following (modified to fit your environment):

```
/usr/bin/mail -s "Good Day!" someemail@address.com  
< /home4/blipdebi/mail.txt
```

This command mails the contents of the text file to the given email address. You have to specify the complete location for the text file, as Cron won't know where it is, otherwise. You have to provide the complete location for the application, too, even if it's one universally available, like *mail*. The location is most likely */usr/bin/mail* for mail. Just assume Cron knows nothing about the system or its applications, and go from there.

To create the Cron job, in the form below the label Add New Cron Job, you'll be providing information about how often to run the job. There are presets available for the most common options, but we're going to walk through the steps to set each value.

There are settings for minute, hour, day, month, and weekday. The Cron job will run every 10 minutes, every hour, and every day. In the first field, you can type in 10, or you can select the option labeled "Every 10 minutes (* / 10)" from the Common Settings dropdown. For the rest of the fields, you're going to type in an asterisk, which symbolizes "every," as in every hour, every day, every month, and so on. Once you've entered the values for the schedule, type the command into the Command field. Your Cron job entry should look like that given in [Figure 6-4](#).

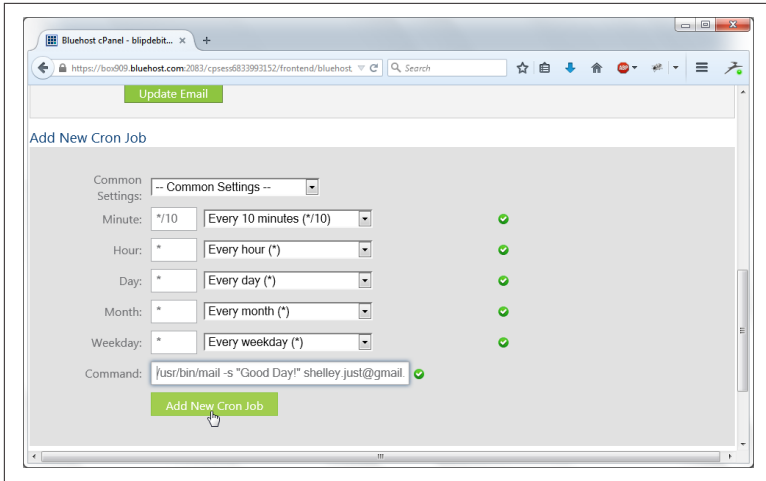


Figure 6-4. Setting up new Cron job

One other option is to have the system send you an email each time the Cron job is run. The email will contain output, if any, from your Cron job. You'll want to provide an email the first time you create a Cron job, because your email will receive any errors that might result from running the job. If no errors occur, and you're running a frequently occurring Cron job, you can delete the email request.

All existing Cron jobs are listed in a table below the form to add a new Cron job. You can delete or edit an existing Cron job via this table.

Command Line Access with SSH

For the most part, everything you need is available via cPanel. The whole purpose for cPanel is to enable all of the functionality you need to manage your website without having to actually log into the server and do it manually.

There may be a time, though, when you'll want to directly access the command line on your server. For instance, I prefer to use the Linux text editor and make simple edits to files directly via SSH, rather than download the file, make the change, and upload it again.

To access the command line, you're going to need the *Secure Shell*, or SSH. SSH is a protocol that enables secure connection between your home computer and your remote server. Another method is to

use *Telnet*, but your data wouldn't be secure. Because of the lack of security, most hosting companies only allow you to use SSH.

You'll use software on your PC to make this connection. Both the Mac and Linux PCs have this capability installed by default. Windows users will need to download the software.



Bluehost SSH Documentation

Bluehost provides excellent documentation for setting up SSH, including how to **establish a connection using a Mac or Linux PC**. Much of what I cover is covered in the Bluehost documentation, but I'll pull the pieces together into a single set of steps.

I'm going to demonstrate creating a SSH connection on Bluehost, but the steps should be very similar to any cPanel based shared host-system that allows SSH access. In addition, I'll demonstrate how to connect to the server using **PuTTY**, the most commonly used Telnet and SSH application for Windows, freely available to all.

To start, download PuTTY. There is an installation program, but you can just download the PuTTY application directly from the **download page**. Place it somewhere you can easily access on your computer.

Next, find the cPanel group labeled Security, and click the icon labeled SSH/Shell Access. In the window that opens, you may get a warning about needing to verify your account before enabling SSH. Since the *blipdebit.com* account was verified when it was first created, we don't have to go through this step. Your experience may vary depending on your host.

Once you have satisfied the preliminary requirements, click the button in the SSH page labeled Manage SSH Access. SSH is disabled by default. The page that opens has a simple drop-down box with two options: to enable or disable SSH. Select the option to enable SSH and click the Submit button, as shown in **Figure 6-5**. That's it: you're now enabled for SSH access. Now you're ready to create your connection.

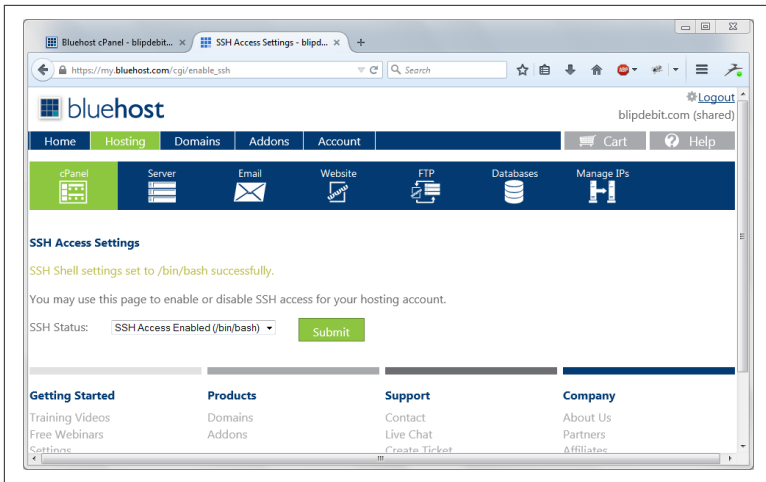


Figure 6-5. Enabling SSH

You don't have to do anything else to be able to connect to SSH at this point. If you have PuTTY installed, you can open the application, type your domain name into the Host Name field in the Session window, and then provide your username and password. You'll be connected to your server.

There's an additional step, though, that can simplify your connectivity and provide better protection for your account password. You can generate a *public/private key pair*: encrypted files that provide the necessary connection information. The public key file lives on the server, and the private key file lives on your computer. Instead of entering your account password, you enter a unique *passphrase* when prompted.

To generate the public/private key pair, access the cPanel SSH page again, and find and click the Manage SSH Keys button. In the page that opens, click the "Generate a New Key" button, and fill in the fields, as shown in Figure 6-6. I opted for the default Key Name when I created my keys, but you can use whatever you wish. Make sure to pick a unique passphrase, and record it somewhere: you will need it. Choose RSA encryption type and select the 2048 Key Size. Finally, click the Generate Key button.

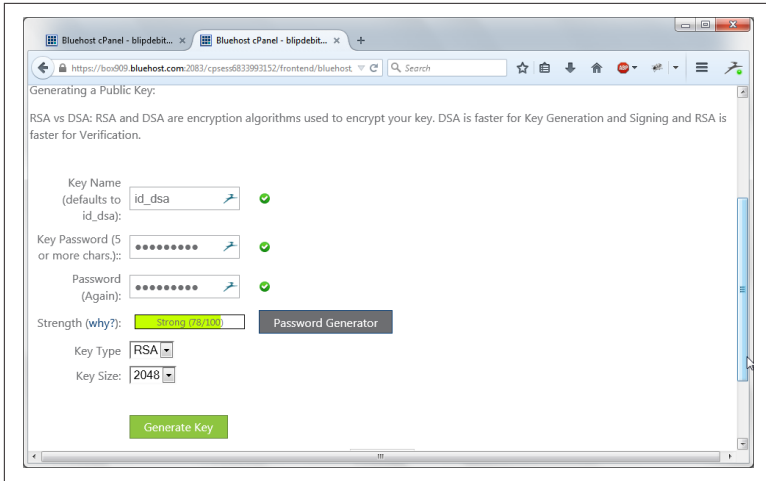


Figure 6-6. Generating the public/private key for SSH

The key appears in a table in the Manage SSH Keys page. It's not currently authorized. To authorize the key, click the Manage Authorization link, and in the page that opens, click the Authorize button. Now you have your authorized public and private keys, as shown in Figure 6-7.

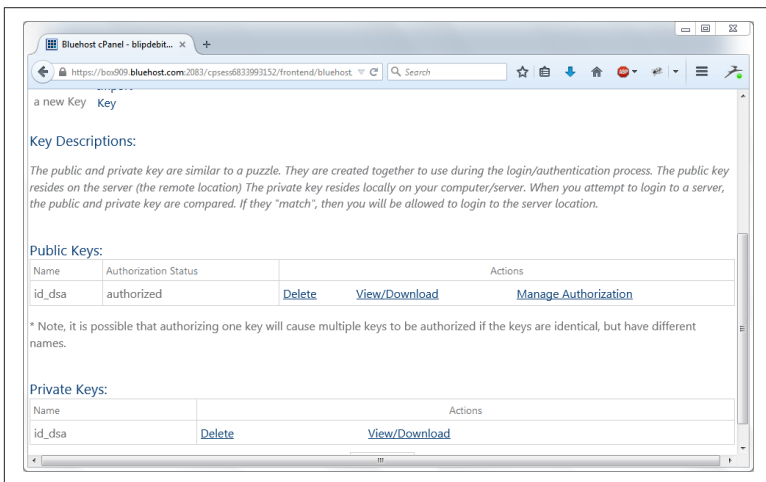


Figure 6-7. Authorized public/private keys

You're going to download the private key to your computer, into a location you can easily find again. Once downloaded, you'll need to generate a PuTTY *ppk* file, using the PuTTYgen tool, downloaded

from the [PuTTY download page](#). Just like PuTTY, the tool is ready to use as soon as you download it: you don't need to run any installation program.

Double-click the PuTTYgen tool to run the application. In the small window that opens, click the Load button. Locate the newly downloaded private key and load it. You'll be prompted to enter your passphrase, as shown in [Figure 6-8](#).

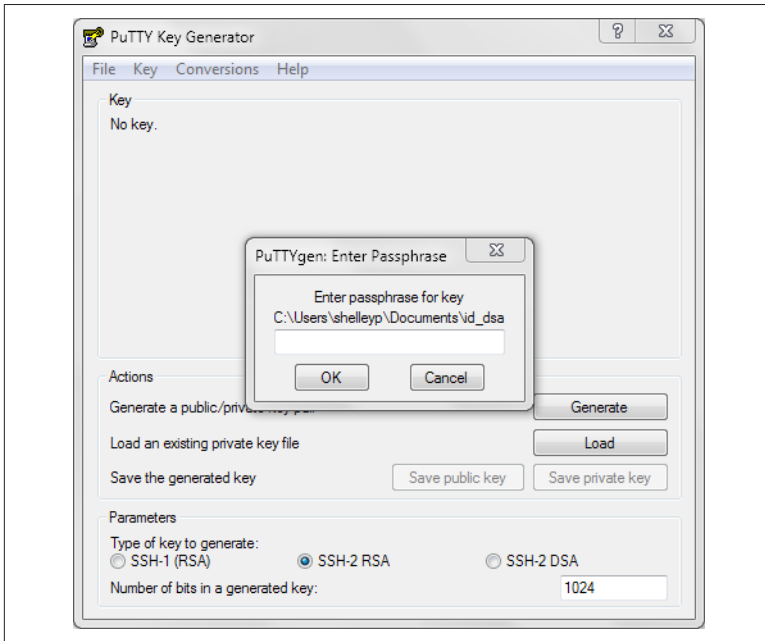


Figure 6-8. Enter a passphrase to generate a ppk file

After the key is loaded, click the “Save private key” button, and save your *ppk* file to a easy to locate location. Close the PuTTYgen application, and open PuTTY.

PuTTY has options on the left and form fields that open on the right. It should open to the Session form field. Type your domain name into the Host Name field, as shown in [Figure 6-9](#).

Click the Data option under Connection in the left, and in the form that opens, type your username into the “Auto-login username” field, as shown in [Figure 6-10](#).

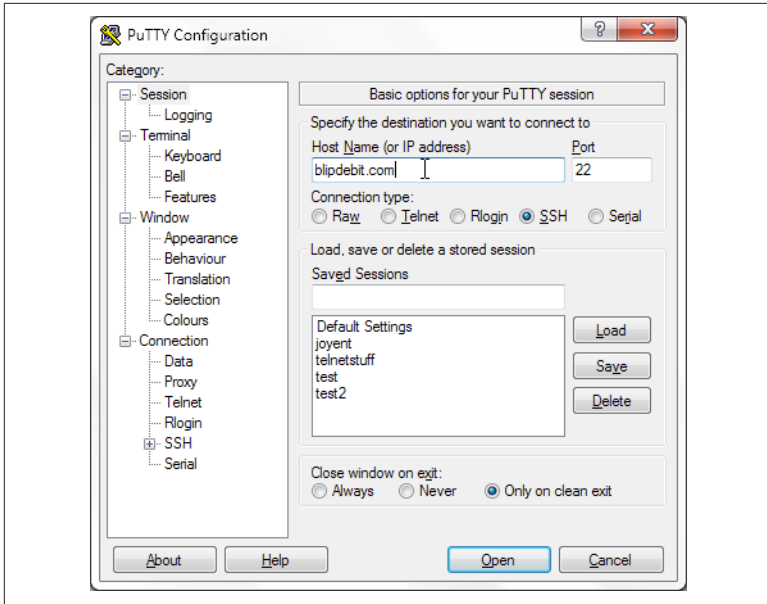


Figure 6-9. Enter domain name into Host Name field

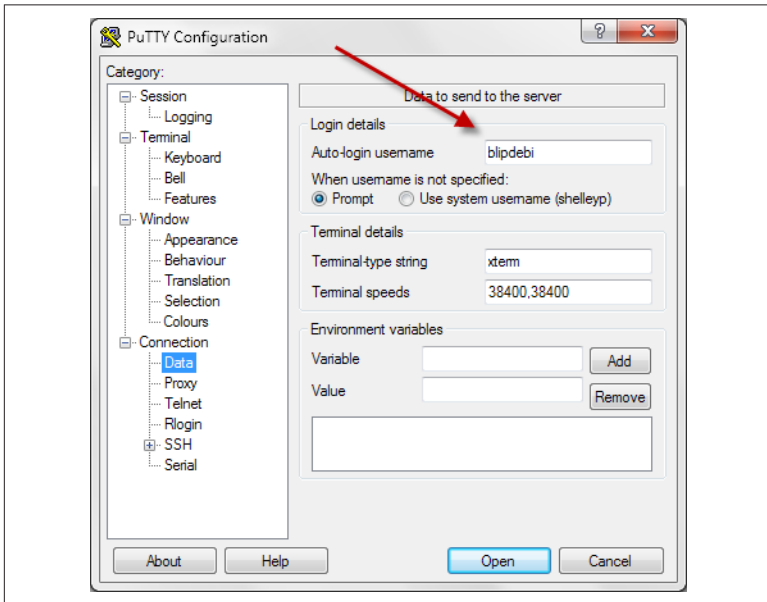
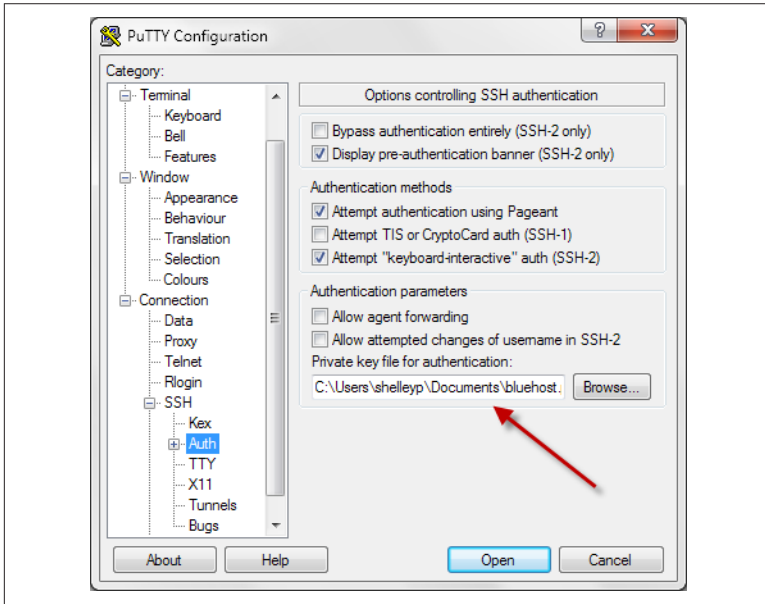


Figure 6-10. Type in username

Next, click the SSH option, and make sure the Preferred SSH protocol version is set to 2. Click the Auth option listed under SSH, and in the form that opens, browse for the private key *ppk* file you just generated, as shown in [Figure 6-11](#).



*Figure 6-11. Find and load the *ppk* file you just generated*

Return to the Session window, type a name into the Saved Sessions field, and click Save to save the session information. Whenever you want to connect, click your saved session, and click Load to load the settings, then Open to open the connection, demonstrated in [Figure 6-12](#).

The first time you connect to your server, you'll get a message similar to the following, except your domain name displays:

```
The authenticity of host 'blipdebit.com' can't be established.  
RSA key fingerprint is (sequence of characters).  
Are you sure you want to continue connecting (yes/no)?
```

Verify the name of the domain, and type **Yes**. Now you're connected.

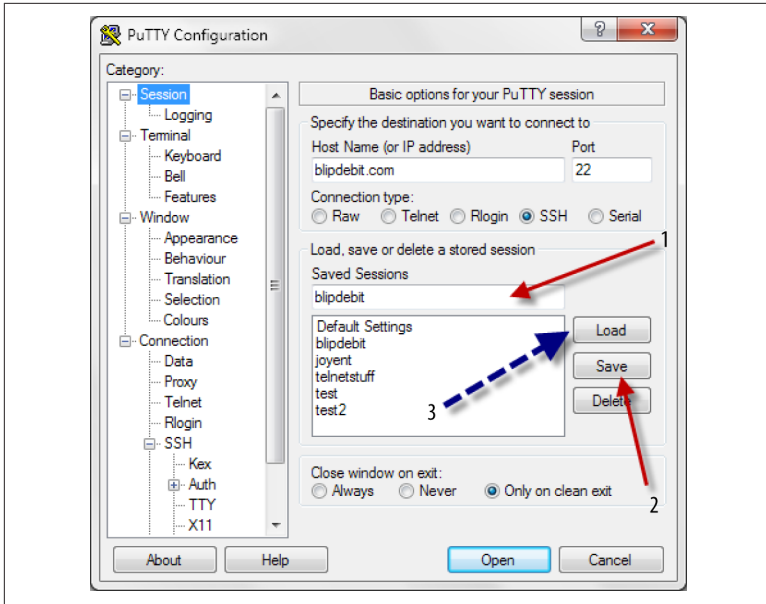


Figure 6-12. Saving the session data and loading it to connect

You'll need to brush up on your Linux command-line skills to do anything in your newly opened terminal connection. If you've not used Linux before, check out the new Linux user resources I mentioned in [“Managing Recurring Tasks with Cron”](#) on page 111. They'll get you started.

Secure FTP (SFTP)

SFTP is a way of transferring files using the security established by SSH. Any commands and transmitted data are encrypted, which means no snoop can peek in and see what you're moving between your server and your local computer.

Once you've enabled support for SSH, you can also use a more secure form of FTP: secure FTP or SFTP. As a matter of fact, once you've enabled SSH access for your website, you won't be able to connect using regular FTP. No worries, though, because connecting with SFTP is a snap.

When connecting with your FTP application, such as Filezilla, select the SFTP protocol option, and make sure the port is set to 22 (the default port for SFTP). You're all set and ready to transfer files.

Website Statistics

Website statistics provide information about how many visitors come to your website(s), what pages they access, where they come from, and even information about the browsers they use. The latter is useful when you're tweaking your website design, but the former—who is visiting what pages, and where they came from—can help you improve the overall impact of your site.

Hosting services that feature cPanel offer multiple statistical programs. You can pick the same program for all sites, all programs for all sites, or a combination.

To set up statistics for your site, access the Statistics group in cPanel. Click the icon labeled Choose Stats. In the page that opens, shown in **Figure 6-13**, a table shows the sites and provides options for the statistic programs, in this case AWStats and Webalizer. For now, check all options for all sites. You can always drop programs if you find you prefer one over the others.

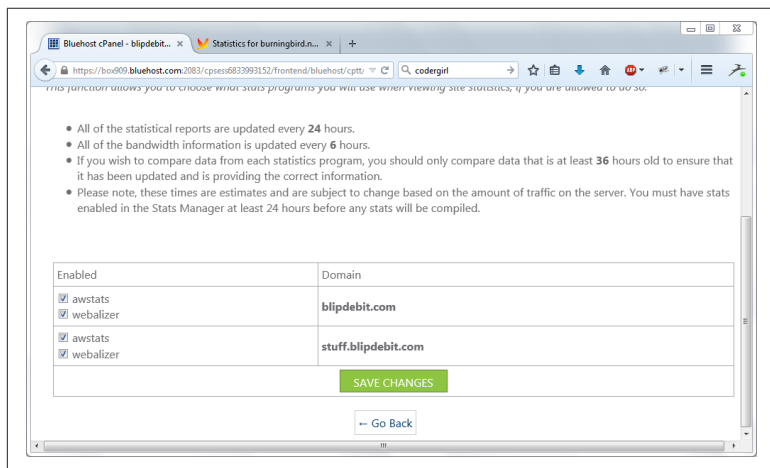


Figure 6-13. Picking statistical programs for each site

It's going to take at least 24 hours before enough information is collected to make the statistics meaningful. And it really takes at least a month, or more, to get a real understanding of visitor patterns.

Once there is enough collected data, there are icons in the cPanel to access the statistical programs. You should have access to AWStats and Webalizer, but this differs based on the hosting company. My

personal favorite is AWStats, which I use at my *burningbird.net* site, as shown in **Figure 6-14**. In the image, we’re looking at the search engine phrases resulting in visits to the site, my personal favorite of all the statistics.

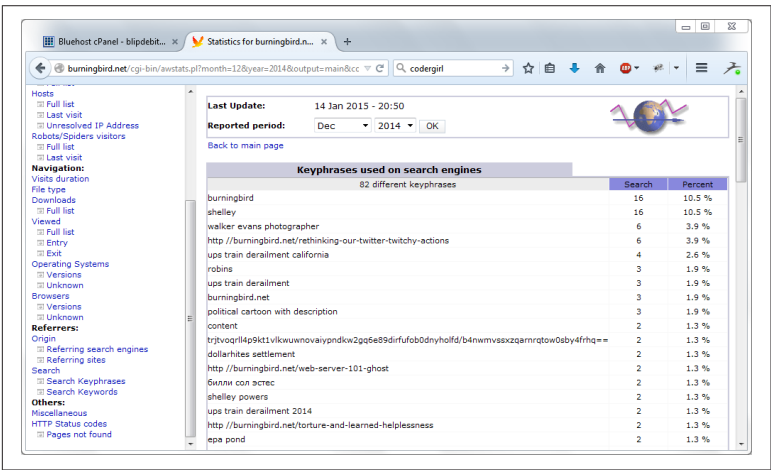


Figure 6-14. Checking out search phrases resulting in site visits

You can also incorporate other statistical and analytical applications. Wordpress has several plugins (including Jetpack) that incorporate statistics directly in the web pages. Other services, such as **Google Analytics**, work by embedding code into each web page that’s tracked (also supported by a Wordpress plugin).

The important point to keep in mind when it comes to statistics is not to get so caught up in it that you spend more time with them than the sites themselves. Site traffic takes time to build, and it shouldn’t be your primary focus. Statistics are nothing more than a useful tool.

Adding Support for Digital Certificates and SSL

For the most part, your web content is accessed via the HTTP protocol. However, there may be circumstances where you want the increased security offered by incorporating support for HTTPS, or *secure HTTP*. Many websites now prefer to serve their web pages using HTTPS because data sent to and from the server is encrypted. Encrypted data is data that can’t be snooped or sniffed, which means

your usernames, passwords, and other sensitive information is protected.

Adding support for HTTPS requires adding support for SSL (Secure Sockets Layer), and this means having access to an SSL certificate. An SSL certificate is a digital file that binds a key with an organization, in such a way that the HTTPS protocol is enabled (over the default port of 443). SSL certificates can be either *self-signed* or signed by a certificate authority (CA). The problem with self-signed SSL certificates is if you use them on your website, every browser that accesses your web page will put up a message basically screaming that your site is unsafe and strongly advocating people not access it. In other words, self-signed certificates are not useful with publicly facing web pages.

Shared hosting companies offer shared SSL certificates with some or all accounts. However, there are restrictions for using the certificate, including traffic limitations and having to use specific URLs. For instance, Bluehost limits files served to be less than 100 KB, or they'll be truncated. And you have to use URLs based on the pattern:

```
https://secure.BlueHost.com/~username
```

where *username* is your own username, such as *blipdebi*. To incorporate into your website, you'd have to add entries to your *.htaccess* file that redirect your website's URLs to the Bluehost URLs, but this type of redirection can conflict with other forms of redirection implemented by your weblogging tool.

The Magical .htaccess File

In **Chapter 5**, we explored the ability to change permalinks for Wordpress pages. Rather than a cryptic URL such as *http://example.com/?p=N*, we can use something more graceful, and meaningful, such as *http://example.com/some-article-story*.

The behind-the-scenes magic that makes this work is the *.htaccess* file. This *hidden* file (the beginning period makes the file hidden, which means it doesn't show up if you list the contents of the file) gives us a place to put directives to the web server—taking the URL that people type in, and doing something with it so that tools can process the request without the individual being aware this is even happening.

For the most part, we don't have to touch the `.htaccess` file. It's modified by the tools we use. However, if you want to learn more about it, check out [.htaccess File Overview](#) from C4learn.com.

The only workable approach for incorporating HTTPS support for your website is to get your own SSL certificate, signed by a CA. Unfortunately, SSL certificates from CA authorities can be pricey. You have to pay an annual fee to use the certificate.

To get your own certificate, you'll first need to get your own, unique IP address. Currently you're sharing an IP address with others, and this isn't going to work if you want to use your own SSL certificate. Shared hosting companies offer individual IP addressed, though you will have to pay an extra fee for the privilege (currently \$3.33 per month in Bluehost).

Hosting companies also, typically, have an arrangement with one or more CAs, making it much simpler to get an SSL certificate. Bluehost works with two companies, with prices ranging from \$49 to \$299 annually, depending on your requirements. If you're not running a storefront, the lower-cost alternative should be all you need. However, if you want to use the certificate with your top-level domain and any subdomains, you will need to purchase a *wildcard certificate*. This is a certificate variation that provides HTTPS support for your main domain (*blipdebit.com*), and all first-level subdomains (such as *stuff.blipdebit.com*).

To use one of the Bluehost-partnered CAs, access the [Addons page](#), get the dedicated IP address, and then select which CA you're interested in. The certificate is then automatically installed. Other hosting companies should have similar procedures in place.

If you're interested in using an SSL certificate from a third-party CA, you'll still need to get your dedicated IP address first. Then you'll need to perform a series of actions. I'll cover those necessary for Bluehost, but the steps should be similar for any shared hosting company.

First you'll need to generate a private key. When you set up your SSH connection, you created a private/public key combination to support this type of connectivity, and you'll need to do something similar for the SSL certificate. For Bluehost users, generate the private key by finding the Security group in your cPanel and then click-

ing the icon labeled SSL/TLS Manager. In the page that opens, click the link labeled “Generate, view, upload, or delete your private key.” In the page that opens, select the key size of 2,048, provide an optional description, and then click the Generate button. The key is displayed in the next page. It’s a good idea to copy the contents of the Encoded Private Key to a file on your computer, just in case you want to use that key in another server.

Once you’ve generated the private key, next you’ll generate a Certificate Signing Request (CSR). You’ll need to provide this to the CA. Returning to the SSL/TLS Manager page, click the link labeled “Generate, view, or delete SSL certificate signing requests.” In the page that opens, complete the form in the page, providing the domain(s), company name, organization name, city, state, country, email, and passphrase. If your certificate isn’t for a company, you can use your domain name for company name, and whatever you wish for organization. Make sure the email address is valid and one you have access to. And don’t use an important passphrase, because it’s stored in an unencrypted format.

If you’re only interested in covering one domain, just give the domain name (e.g., *blipdebit.com*). If you’re interested in covering all your first-level subdomains (e.g., *stuff.blipdebit.com*), you can specify a wildcard domain, demonstrated in [Figure 6-15](#). The certificate would cover your domain and subdomains. However, CAs charge more, sometimes significantly more, for wildcard certificates.

When you submit the form, the CSR is generated. You’ll need to copy the encoded certificate signing request into a file on your PC. This is the file you’ll have to submit to the CA when you purchase your SSL certificate.

The process to purchase the actual certificate does vary by CA. Each should provide sufficient instructions to complete the act. Once you have your certificate (file with extension of *.crt*), you’ll load it to your hosting company’s server. In the case of Bluehost, access the SSL/TLS Manager page one more time, and click the link labeled “Generate, view, upload, or delete SSL certificates.” In the page that opens, click the button to browse for, and upload the certificate. Once uploaded, you’ll need to open a ticket to request that your key and certificate be installed on the server.

Bluehost cPanel - blipdebit... x +

https://bo009.bluehost.com:2083/cpsess5607859809/frontend/bluehost/ssl/c Search

Generate a New Certificate Signing Request (CSR)

Use this form to generate a new certificate signing request for your domain. Your SSL certificate authority (CA) will ask for a certificate signing request to complete the certificate purchase. Your CA may require specific information in the form below. Check with the CA's CSR requirements for the Apache web server.

Key Required

Generate a new 2,048 bit key.

Domains Required

*.blipdebit.com

Provide the FQDNs that you are trying to secure, one per line. You may use a wildcard domain by adding an asterisk in a domain name in the form: *.sample.com. NOTE: Many CAs charge a higher price to issue multiple-domain certificates (sometimes called "UCCs" or "SAN certificates") and certificates that include wildcard domains.

City Required

Saint Louis

Figure 6-15. Inputting wildcard domain into CSR form

It is simpler to just get your certificate through your host, but the price difference can be significant. For example, purchasing a wildcard certificate from Comodo via Bluehost costs \$299 a year. Purchasing the same wildcard certificate from Comodo using the registrar we used for our domain name, Namecheap, costs \$94 a year. I would say the cost difference makes up for the additional complexity.

Hope for Affordable (as in Free) SSL Certificates

If you're experiencing price shock at the cost of SSL certificates, have hope: an affordable (as in free) SSL certificate CA should be opening for business later in 2015.

The Electronic Frontier Foundation, Mozilla, Cisco, and other organizations have banded together to bring us **Let's Encrypt**, a CA providing uncomplicated, freely available SSL certificates.

This is a game changer and desperately needed. Many companies are encouraging everyone to migrate their pages to HTTPS, but the cost and complexity of SSL certificates has been a road block. Starting in 2015, hopefully the roadblock will come to an end.

Read more about Let's Encrypt at the **EFF**.

So you have your certificate installed. Now what? You can force all of your web pages to be served as HTTPS with a couple of relatively minor modifications.

For your static pages, force them to be served as HTTPS by making a small modification to your site's *.htaccess* file. Use the Code Editor in the File Manager (covered earlier) to add the following to this file:

```
RewriteEngine On
RewriteCond %{SERVER_PORT} 80
RewriteRule ^(.*)$ https://yourdomain.com/$1 [R=301,L]
```

If you want to serve your Wordpress pages as HTTPS, there is a plugin for that. Actually there are several, but one of the more popular is the **Wordpress HTTPS plugin**. Just follow the plugin installation instructions.

Moving Your Site

At some point in time you may need to move your website to a different server/host. How complex the move is depends on what you have installed. If your site consists of static pages, and something like a Wordpress weblog, you'll need to copy all of your files, but you'll also need to copy your weblog's database.

Even if you're not moving your website, it's a good idea to make your own backup of all your files, at least once a month. Yes, your shared host also does backups, but there's no such thing as bad redundancy when it comes to web content.

Backing Up Your Files and Moving Them

You can easily copy all of your file using your FTP software. Before you do so, though, make sure that the FTP tool you use is set to display hidden files, such as *.htaccess*. If you don't, these essential files won't be copied with the rest.

Create a folder in your computer to contain your backup files. In the FTP tool, such as Filezilla, copy the files by dragging them from your server to your PC, as shown in **Figure 6-16**.

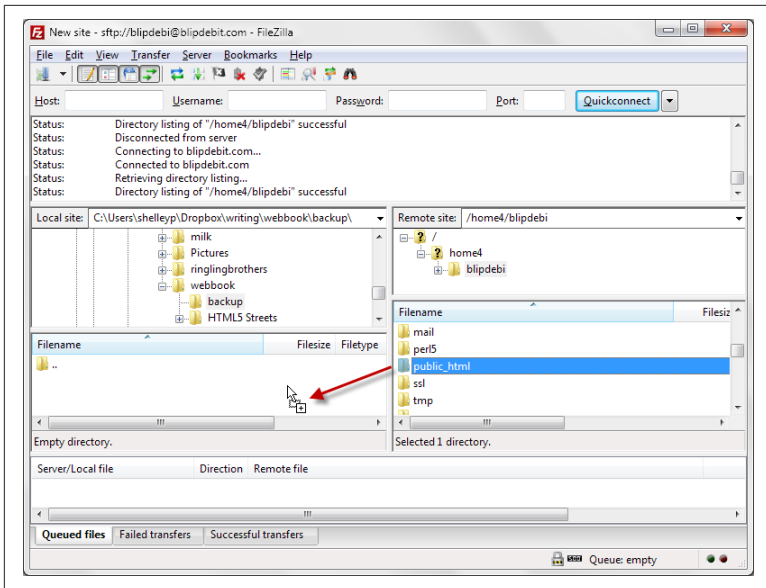


Figure 6-16. Downloading the `public_html` files

The `public_html` subdirectory is the location of your publicly accessible files and contains the files you want to copy. You shouldn't need to copy files in any higher-level subdirectory, unless you placed the files there yourself. The directories are used by the server for log files, email storage, and so on.

Once you have a copy of all your files, use the FTP client to relocate them in your new server. If you're moving hosts, you'll want to move things as is. This includes keeping the exact same subdomains. Keep the process as simple as possible. You don't want to move hosts *and* rearrange your site at the same time.

Exporting and Importing the Database

If you have installed a CMS like Wordpress, you'll need to export the database for the tool, as well as copy the files. You can export the database using phpMyAdmin, which is available as an icon under the Database Tools group in cPanel. After logging into phpMyAdmin, click the Databases icon in the top menu bar, and select the database you're exporting. If you have more than one database, you'll need to export all of them.



Prepare Your Weblog for the Move

Before you begin the process of exporting your weblog's database, you should put your site into *maintenance mode*. This is a page that tells people that the site is currently undergoing maintenance and will be back shortly. If you think there's a plugin to manage this for you, you'd be right: I suggest you use **WP Maintenance Mode**.

When you click the database, the tables display, similar to that shown in **Figure 6-17**. Along the top are menu options, including one for exporting the database. Click it.

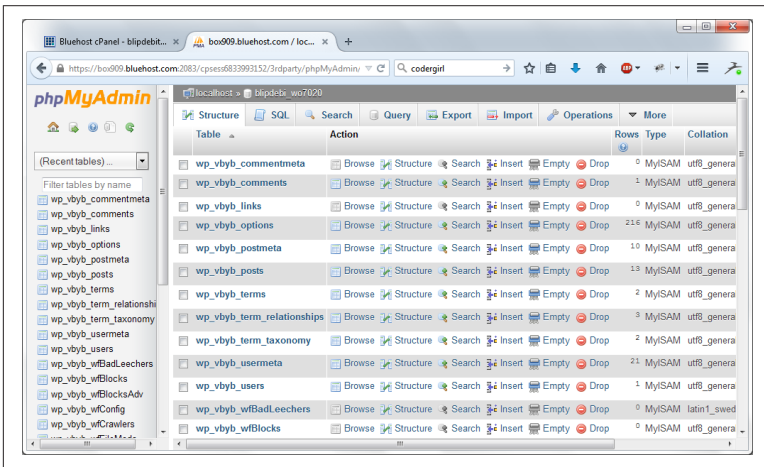


Figure 6-17. The Wordpress database

There are two options for exporting the database: Quick or Custom. Keep it simple and pick the Quick option. Make sure the format is SQL, and don't check the option to store the file on the server: you want to store it on your computer, as shown in **Figure 6-18**.

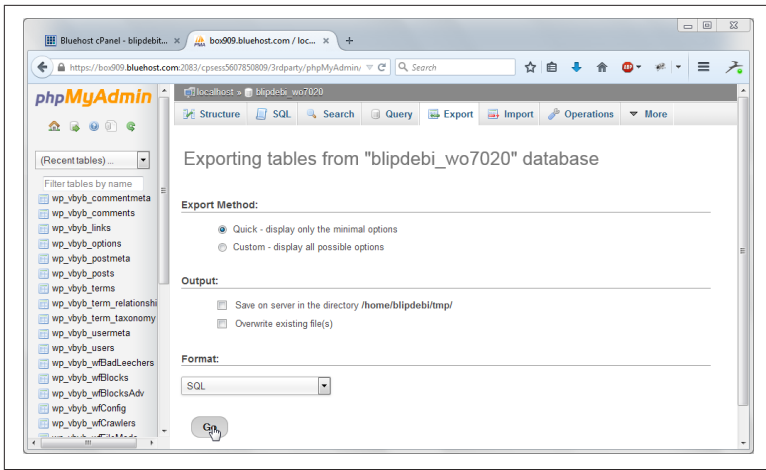


Figure 6-18. About to export the database

You now have a copy of the database. To move it to your new host, you'll need to create the database and import the files. You can't create the database using phpMyAdmin, you'll have to use the MySQL Databases tool. Like phpMyAdmin, it should be listed in the cPanel Database Tools group.

In the page that opens, there's a space to input the name for the database. You'd like to be able to keep the database names the same, but frequently, hosting companies annotate database names, adding their own imprint, as shown in **Figure 6-19**. Not a problem: there's just a file we'll need to tweak after the database is finished.

You'll also need to create a user. Ideally, the user would have the same username and password created when you first created the weblog. However, again, the username may be annotated with the hosting company's imprint, so type in as close a name as you can. Give the user *all* privileges when prompted. Once you create the user, add them to the database. All three activities—creating the database, creating the user, and adding the user to the database—are performed in the same MySQL Databases page.

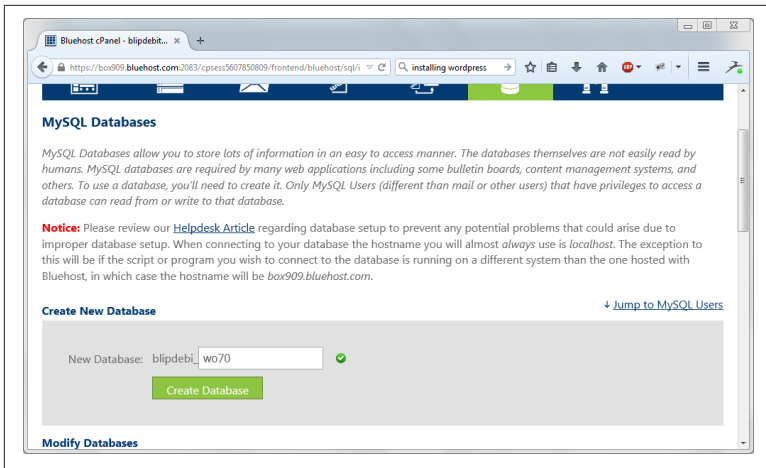


Figure 6-19. Creating a new database

Once the database and user are created, then you can use phpMyAdmin to import your exported SQL into the new database. Open phpMyAdmin, and click the newly created database to open it. Once opened, click the Import button in the top menu bar. Browse for and select your exported database SQL, leave the other fields in the page at their default, as shown in Figure 6-20, and then click the Go button. Once it's imported, you should have a duplicate of your previous database.

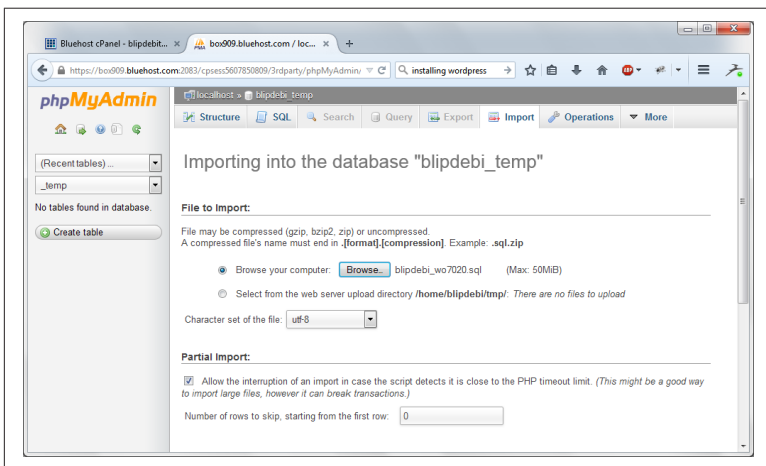


Figure 6-20. Importing the database entries into new database

The only other change you might have to make is if the database and/or username have changed. If this happens, you'll have to edit the *wp-config.php* file in your Wordpress weblog directory. You can edit it using the Code Editor, one of the tools available in the File Manager. **Figure 6-21** shows the file opened in the Code Editor, and you can easily spot the two values that need to be changed: `DB_NAME` and `DB_USER`. If the password is different, make sure to change the value for that, too.

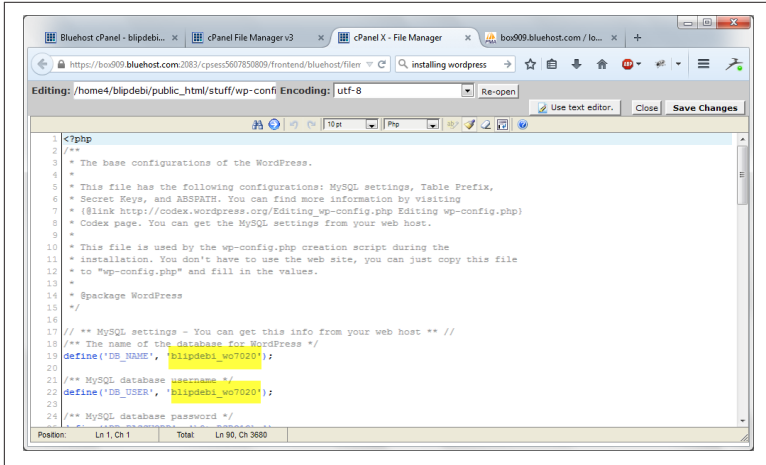


Figure 6-21. Editing the wp-config.php file

Change the database and/or username, and save the file. Open the weblog, and you should be ready to go.

If you're concerned about the process, you can do a trial run. Create a new database and username in your existing host, add the username to the database, and import the exported SQL into it. Then change the *wp-config.php* file and see if you have any problems with the weblog using the newly created database. If you don't have any problems with the new database on the existing server, you shouldn't have any problems with the database on the new server.

About the Author

Shelley Powers has been working with and writing about web technologies—from the first release of JavaScript to the latest graphics and design tools—for more than 12 years. Her recent O'Reilly books have covered the semantic web, Ajax, JavaScript, and web graphics. She's an avid amateur photographer and web development aficionado who enjoys applying her latest experiments on her many websites.