CHAPTER 6

CREATING LINKS WITH ANCHORS

Up to this point we've focused on creating well-structured web pages, marking up our content using the right tag for the job. We've introduced you to most of the tags you'll need to include text and images on your web pages. If you've been following along with the homework, you should by now be able to create quite complex web pages using structured markup. All good, but there's something distinctly lacking: **links**.

The Web is all about links; without them there would be no Web. By now you're aware that we're using HyperText Markup Language, but we've yet to create some actual **hypertext** to link our separate pages together. Have no fear; this is the chapter where we introduce you to links to enable you to do this.

The first two initials of HTML stand for HyperText—one word, two initials (because HML wouldn't sound as snappy). The *New Oxford American Dictionary* (Oxford University Press, 2005) states that hypertext "links topics on the screen to related information and graphics, which are typically accessed by a point-and-click method." This chapter introduces you to the point-and-click aspect of web design, or to be more specific, **anchors**.

What is an anchor? Simply put, an anchor is a means of tying together separate pieces of information. We can use anchors to link to other pages or resources on the Web, to other web pages within our web site, and even to different sections of the same web page, which is especially useful if we're creating a web page with a lot of information.

But how do we include anchors on our web pages? The answer is simple: we use the <a> tag, also known as the **anchor** tag, and its necessary attributes, which we introduce next. This chapter will enable you to link your carefully crafted web pages together to create a web site. Let's get started.

Meet <a>

What makes links important? The answer is that without hypertext links the Web wouldn't be the Web, it would simply be a collection of separate, unconnected pages. The W3C states the following:

> *HTML offers many of the conventional publishing idioms for rich text and structured documents, but what separates it from most other markup languages is its features for hypertext and interactive documents. Although a simple concept, the link has been one of the primary forces driving the success of the Web.*
>
> `http://www.w3.org/TR/html401/struct/links.html`

Links point the browser to a destination anchor, which can be any form of web resource: for example, an HTML document, a specific part of an HTML document, a link to an e-mail address, or even an image, a video clip, or a sound file. The following example contains a link in its simplest form: it has **link text**, in this case the word *Cheeta*, and a **destination anchor**, which is an HTML document named `cheeta.html` located at the domain `www.famousprimates.com`:

```
<a href="http://www.famousprimates.com/cheeta.html">Cheeta</a>
```

The a element instructs the browser we're linking to another piece of information, and the `href` attribute informs the browser of the location of that information. Without the `href` attribute, the browser wouldn't know where to look for the information we are linking to. By now you won't be surprised to learn that `href` stands for **hypertext reference**. It should also come as no surprise that the contents of the `href` attribute point to, or refer to, the address of the resource we're linking to.

Here is the preceding link again, but this time shown within a short paragraph of text to demonstrate how a browser differentiates ordinary, nonlink text from link text:

```
<p>Tarzan's sidekick,
<a href="http://www.famousprimates.com/cheeta_75.html">Cheeta</a>,
celebrates his 75th birthday, as he retires from the movie business.
</p>
```

The preceding example renders in a browser unstyled as in Figure 6-1.

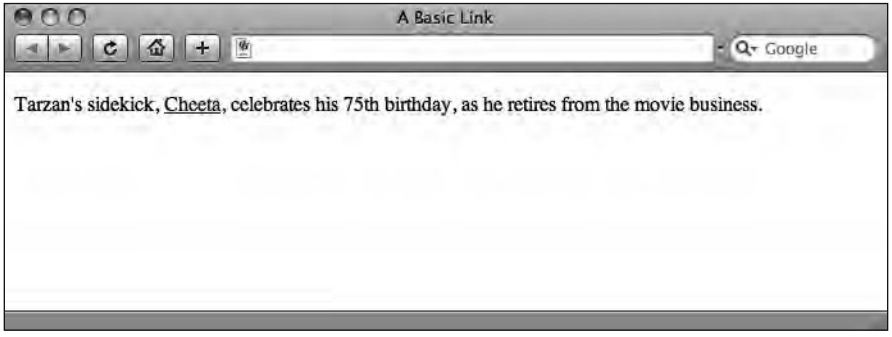

Figure 6-1. Our example link as it renders unstyled in a browser

As you can see, the default style for links in a visual browser is usually blue text with an underline. (As this is a book printed in black and white, you'll have to take our word that the link is, in fact, blue. Better yet, you can take a look at the page in color on the book's companion web site: `www.webstandardistas.com/06/unstyled_links.html`.) However, as we'll cover in Chapters 9 to 12, when we're adding style with CSS, we can change these defaults using a style sheet, switching off the underlining and changing the color of the link, should we wish to do so. We'll introduce you to how to do this in the second half of the book where we focus on CSS.

Using descriptive link text

As you saw in Figure 6-1, everything contained within the opening `<a>` and closing `</a>` tags is highlighted within the browser as a link, in this case the word *Cheeta*.

Our link text can be as long or as short as we like. However, it's good practice to use descriptive language when writing link text so that the user knows what to expect before

clicking the link. Writing good link text is an important, yet often overlooked, part of the design process. Taking some time to write descriptive link text results in a more user-friendly site, something you should strive for.

Take a look at the following two examples. The first example doesn't give the user any indication of what they will find if they click the link, as the word *Cheeta* on its own doesn't offer much in the way of clues. It might link to a biography page about Cheeta, an image of Cheeta, or Cheeta's Wikipedia entry.

```
<p>Tarzan's sidekick,
<a href="http://www.famousprimates.com/cheeta.html">Cheeta</a>,
celebrates his 75th birthday, as he retires from the movie business.
</p>
```

The second example, however, uses language that clearly indicates what the user might expect to find at the link's destination, suggesting a page with a gallery of photos of Cheeta celebrating his momentous birthday.

```
<p>Tarzan's sidekick,
<a href="http://www.famousprimates.com/cheeta.html">Cheeta,
here seen celebrating his 75th birthday</a>, as he retires from the
movie business.</p>
```

Descriptive link text also holds significant weight in search engine rankings and is more valuable for search engines than generic phrases like "click here," for example.

Using descriptive link text is also important when the link might break with a user's expectations, for example, a link to a Microsoft Word document or another file type that might not necessarily open within a browser or might launch another piece of software.

Lastly, it's good practice to use clear link text when linking to large files, for example, a movie file. It's helpful to indicate the size of large files as in the following example so that a user has an indication of what they're committing to before clicking the link:

```
<p><a href="http://www.famousprimates.com/cheeta.mov">Watch Cheeta
as he relaxes during his retirement [QuickTime Movie - 8.4 MB]</a>,
on the beach in Florida.</p>
```

Another way of providing more detailed information about links is through the use of a `title` attribute, which we introduce next.

The title attribute

As with the `<img />` tag introduced in the last chapter, the `<a>` tag has a number of attributes that enable it to work its magic. As you now know, the bare minimum information we need to provide within an anchor is an `href` attribute, which informs the browser where the information we'd like to link to is located on the Web. We can provide a little more detail about the link, however, with the inclusion of a well-written and descriptive `title` attribute.

You may recall you met the `title` attribute in the previous chapter when we introduced adding images. In fact the `title` attribute can be used with all HTML elements except for a select few; however, it isn't strictly required for any. That this attribute can be used with so many elements—and isn't always—is probably one reason to explain why it's less than clear when to use it and what it's for. The W3C states that a `title` attribute is meant to offer "advisory information about the element for which it is set" (`http://www.w3.org/TR/html4/struct/global.html#h-7.4.3`). However, that advice is at best a little vague.

We established in the last chapter that the `title` attribute wasn't strictly required for our `img` elements and that, if you needed to add additional information to images, it might be better to include this through the use of a caption in a `p` element. With links you face a similar conundrum.

First and foremost, it's important to write descriptive link text; however, a well-written `title` attribute can also provide additional information to the user before they click a link that takes them to a destination they might otherwise not want to go to.

The following example expands on our link about Cheeta's retirement party to include a `title` attribute that gives the user a little more information about the link before they click it:

```
<p>Tarzan's sidekick,
<a href="http://www.famousprimates.com/cheeta_75.html"
title="Cheeta's 75th Birthday Party Photo Album">Cheeta,
here seen celebrating his 75th birthday</a>, as he retires
from the movie business.</p>
```

Although well-written link text is of more importance when creating links, the `title` attribute allows for the provision of additional information. In the preceding example, the `title` attribute clearly indicates the user will be taken to a photo album, information that isn't contained in the link text, but information that's useful nonetheless. The information contained within the `title` attribute will usually appear within a tooltip when the user mouses over the link as shown in Figure 6-2.

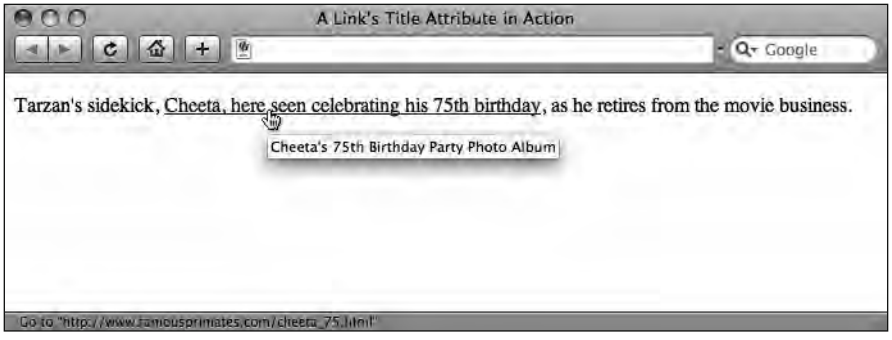

Figure 6-2. A link's `title` attribute in action as the user mouses over a link

Another important aspect of using the `title` attribute when creating links is that screen readers for the visually impaired can be set up to read out its contents; however, this is

sometimes at the expense of the link text contained within the <a> tags. If you are designing specifically for users with accessibility issues, we strongly recommend a little further reading on this thorny topic. We've provided some links to get you started at the book's companion web site:

```
www.webstandardistas.com/resources
```

Let's create some links!

We mentioned briefly that links can be used to link to a variety of destinations. We now introduce you to a variety of examples: **external links** (to other web sites or resources on the Web), **local links** (to other pages within your own web site), **internal links** (links to specific areas within a web page itself), and, lastly, links to other types of resources (an e-mail link, for example).

In this section we introduce a number of different ways of linking to other resources online, showing you how to add links to your own web pages. Good news, we're about to start creating a properly linked-up web site.

External links

Now that you're familiar with the component parts of a link, you know that the address of the page we want to link to is contained within the `href` attribute. Let's take a look at creating a link to another web site *elsewhere on the Web* on one of our web pages.

> *From this point onward we'll be creating our link examples without* `title` *attributes. This is to ensure our markup is easier to read.*

If we wanted to link to Wikipedia's monkey page, for example, we can do this by adding the following markup to our web page:

```
<a href="http://www.wikipedia.org/wiki/monkeys">Wikipedia's
monkey page</a>
```

The `href` attribute in this example contains the full address, or Uniform Resource Locator (URL), of the web site we'd like to link to. Note that the link's address starts with `http://`—this informs the browser to look for this link on the Web (using the HyperText Transfer Protocol). For external links to other web sites *that we're linking to*, the `http://` part is *essential to include*; without it the link will not work.

The preceding example is useful to give you an idea of what one particular example of a link looks like, but let's take it a bit further by creating a list of external links. You can use this to form the basis for a links page for the web site about famous monkeys and apes that you're building as part of the homework.

We'll link to a variety of files, including those for an image, a video, and a web page. Although this adds to the complexity, this will give you an idea of what a typical list of links looks like. While it might appear a little daunting at first glance, there's nothing in the following example that you haven't been introduced to. We're using an unordered list, or `ul` element, to give our list some added structure.

```
<h1>Famous Online Primates</h1>
<ul>
  <li><a href="http://www.explorepahistory.com/images/➥
  ExplorePAHistory-a0h2z2-a_349.jpg">
  Cheeta - Tarzan's Sidekick</a></li>
  <li><a href="http://www.youtube.com/watch?v= w3TwHW8ItAA">
  Son of Kong fights Giant Bear</a></li>
  <li><a href="http://classicgaming.gamespy.com/➥
  View.php?view=GameMuseum.Detail&amp;id=297">Donkey Kong -
  The Original Gaming Ape</a></li>
</ul>
```

You can see a more complete version of this example that demonstrates a number of external links in action on the following page at the book's companion web site:


```
www.webstandardistas.com/06/external_links.html
```

The dreaded ampersand and the validator

When validating your pages, you might at some point encounter an issue relating to URLs in links containing an ampersand (&) character. Including URLs containing ampersands in your links will lead to a frustrating and somewhat obscure error.

The reason is that the ampersand is a **reserved character** in XHTML and therefore needs to be **encoded**—replacing & with `&`—in order for your page to validate. As a consequence of this, the following markup is not valid:

```
<a href="http://www.webstandardistas.com/the_dreaded_&.html">
The Dreaded Ampersand</a>
```

This markup returns the following error message from the W3C Markup Validation Service: "You used an unescaped ampersand '&': this may be valid in some contexts, but it is recommended to use '`&`', which is always safe."

You'd be forgiven for wondering what this error message means—and whether or not ampersands are "safe" to use in URLs is another matter; however, if we replace the & with `&` in the preceding link, the page will validate:

```
<a href="http://www.webstandardistas.com/the_dreaded_&amp;.html">
The Dreaded Ampersand</a>
```

It's worth noting that you only need to encode the ampersand within the markup of your page. If you want to type the URL into the address bar of a browser, you should use the unencoded &.

The ampersand isn't the only character you need to encode when writing your markup. You also need to encode < with `<` and > with `>` when *not used as part of an HTML tag*. In short, wherever you use these characters when you're *not* creating tags, use their encoded versions as in the following examples.

The following will not validate:

```
<p>Cheeta & Bonzo Combined < King Kong</p>
```

The following, where the & and < are encoded, will validate:

```
<p>Cheeta &amp; Bonzo Combined &lt; King Kong</p>
```

With the < and > characters it's not just a case of passing validation, as some browsers might incorrectly interpret these as the start or end of an actual tag, which might cause rendering issues.

Forgive us for this brief interlude on the intricacies of reserved characters and character entities—now back to business.

Checking your links

Before going live with your web site, it's important to check that all of your links actually work. A simple error when typing in a link—a spelling mistake or a missing piece of punctuation, for example—can easily result in a link not working, which not only proves frustrating to your users, but is also a little embarrassing for the budding Web Standardista.

There are a number of free web-based tools that automatically check all the links on your web site or check links on a per-page basis. Our friends at the W3C, who provide the W3C Markup Validation Service, also provide one such free link-checking tool. We've provided some links to automated link checkers, including the W3C's, at the book's companion web site:

```
www.webstandardistas.com/resources
```

The (evil) target attribute

You'll see a number of resources online recommending the use of the `target` attribute, which can be used to open links to other web pages in a new window as in the following example:

```
<a href="http://www.wikipedia.org/monkeys" target="_blank">
Wikipedia's Monkey Page</a>
```

Including a `target="_blank"` attribute opens a new (blank) window and loads the page you are linking to into it, leaving your web site still open in a window beneath the new window.

Don't be tempted to use the `target` attribute. Not only is it forbidden in XHTML, but its use is also bad practice as it takes control away from your users. A better approach is to respect your users' wishes and allow them to decide how they'd like to access the links they click.

As tabbed browser interfaces have become increasingly commonplace, users will often choose to open your link in a new tab instead of within the current window. As a consequence, tabbed browsing is to an increasing extent rendering `target="_blank"` null and void, which is no bad thing.

Local links

Now it's starting to get interesting. Having a web page with external links is fine, but a little bit limited. What we really want are a few pages *of our own* all linked together. In other words, a web site of our own.

Imagine we've created a web page titled "Monkeys in Space" to link to all our space monkey pages, and we've named this HTML file `monkeys_in_space.html`. We've also created another web page that we've named `gordo.html` with specific information on Gordo. (Does this ring a bell?) To create a link from the Monkeys in Space general overview page to the page specifically about Gordo, we can use the following markup:

```
<a href="gordo.html">All you ever needed to know about Gordo,
one of the first monkeys in space.</a>
```

Assuming both web pages are stored in the same folder or directory, you can see creating the link is as simple as putting the file name of the file you'd like to link to in the `href` attribute. Note that a local link *does not* contain either the `http://` or the full domain name, just the name of the file linked to.

> *We will cover how to link to pages and resources that are not all in the same folder in the "Linking between different folders in our site" section in this chapter.*

You can see a more complete version of the preceding example that demonstrates a number of local links in action on the following page at the book's companion web site:

```
www.webstandardistas.com/06/local_links.html
```

Use View Source to get a feel for how the links work to link the separate pages together.

Internal links

Although most commonly used for linking *different* web pages together, the a element can also be used to point to specific sections *within a current document or web page*. This can be useful if you've created a long web page with a lot of related information on it.

For example, imagine we've created a long page titled "Apes in the Movies." We might have a section on King Kong, another section on Cheeta, and yet another section on Cornelius. As you can imagine, by the time we include biographical details on our various ape thespians, this page might become quite long.

Using well-structured markup, we've headed the page with an h1 element and headed each of the ape's separate sections with h2 elements. If we give each of these h2 elements an id attribute—a unique identifier that distinguishes each h2 from the other h2 elements on the page—we can link to them within the page's introductory text using internal links, as in the following example, which we explain in full afterward:

```
<h1>Apes in the Movies</h1>
<p>Many famous apes have been featured in the movies.
Some of the most noted include: <a href="#king_kong">King Kong</a>,
<a href="#cheeta">Cheeta</a> and <a href="#cornelius">Cornelius</a>.
</p>
<!-- Imagine a lot of extra content here. -->
<h2 id="king_kong">King Kong</h2>
<p>King Kong is the name of a fictional giant ape from the legendary
Skull Island, who has appeared in several works since 1933...</p>
<!-- Imagine a lengthy section on King Kong here. -->
<h2 id="cheeta">Cheeta</h2>
<p>While inextricably associated in the public mind with
Tarzan, Cheeta as a character was a product of the movies,
never appearing in any of the original <cite>Tarzan</cite> novels by
Edgar Rice Burroughs...</p>
<!-- Imagine a lengthy section on Cheeta here. -->
<h2 id="cornelius">Cornelius</h2>
<p>Cornelius is a chimpanzee archaeologist and historian, appearing
in the original novel of <cite>Planet of the Apes</cite>, and also the
first three installments of the classic movie series of the same
name, from the 1960s and 1970s.</p>
<!-- Imagine a lengthy section on Cornelius here. -->
```

To denote that the links in the first paragraph of the page are internal links, that is, links to another point *on the page we're currently on*, the href attributes start with a # (hash) sign—followed by the section's unique identifier as indicated by its id attribute: in this case either #king_kong, #cheeta, or #cornelius.

These links will cause the browser to jump down to our King Kong, Cheeta, and Cornelius sections when we click their corresponding links within the opening paragraph. This is thanks to the unique identifier, or id attribute, we've added to the opening <h2> tags at the start of each section.

We can give *any* element on the page a unique identifier, or id attribute, allowing us the flexibility to jump from one section of the page to another. Among other things this allows us to create a link at the base of a long page of content that takes the user back to the top of the page as in the following example for the Cheeta page:

```
<h1 id="cheeta">Cheeta</h1>
<!-- Imagine a lot of extra content here. -->
<a href="#cheeta">Back to Top</a>
```

You can see this in action by using View Source at the book's companion web site:

```
www.webstandardistas.com/06/back_to_top.html
```

We can also combine internal links with local links, enabling us to link from one web page to a specific subsection of another web page within our site. Imagine we're on the Cheeta web page, but we want to create a link to the King Kong *section* of the Apes in the Movies web page. We can use the following markup to achieve this:

```
<a href="apes_in_the_movies.html#king_kong">Find out more
about King Kong.</a>
```


What this markup does is first tell the browser to go to the Apes in the Movies web page (apes_in_the_movies.html), and then look for the section on that page with the id of king_kong.

Without getting overly complicated—we'll cover the id attribute in greater depth in Chapter 10—each id is a unique identifier meaning that there's only one element on the Apes in the Movies web page with an id of king_kong. The use of unique identifiers allows us, among other things, to create this internal link.

You can see a more complete version of the preceding example that demonstrates internal links in action on the following page at the book's companion web site:

```
www.webstandardistas.com/06/internal_links.html
```

E-mail links

You've built your web site and you're happy with it, but you'd like some way for others who browse the site to contact you. One way to achieve this is with a link to your e-mail address. To create an e-mail link is simple, as shown in the following example:

```
<a href="mailto:primates@famousprimates.com">Get in touch
with the primates who run this site.</a>
```

As you can see, the content of this href attribute is slightly different from those that you've encountered so far. Instead of starting with http://, the e-mail link starts with mailto: followed by an e-mail address. When the user clicks this link, their e-mail application is launched with the e-mail address specified in the href attribute already entered in the To: field of the e-mail.

> *Spam alert! Putting your e-mail address on your web site can lead to a huge quantity of spam as spambots surf the Web looking for e-mail addresses to add to their databases.*

If you need to insert your e-mail address into a web page, a better approach would be to use a service like Hivelogic's Enkoder, a web-based application that converts your e-mail links into encrypted JavaScript code, which hides them from e-mail-harvesting robots while revealing them to real people. You can read more about the Enkoder at the book's companion web site:

```
www.webstandardistas.com/tools
```

Although JavaScript is beyond the scope of this book, if you've been following along with the homework, you should know enough by now to understand how to embed an "enkoded" e-mail link in your web site after you've looked at the preceding page. It's worth pointing out, however, that e-mail-harvesting robots are getting smarter by the day, and the only guaranteed way to ensure that your e-mail address does not end up in their claws is to not include your e-mail address on the page in the first place.

Another approach for allowing users to contact you would be to include a contact form on your web page. This requires a server-side script to parse the data entered on the form and e-mail it to you, a topic that's a little beyond the scope of this book. However, we wouldn't want to leave you feeling short-changed, so we recommend some web-based form creation tools at the book's companion web site:

```
www.webstandardistas.com/tools
```

Wrapping up

We've covered four types of links in this section:

- **An external link**: `<a href="http://www.wikipedia.org/monkeys">`
- **A local link**: `<a href="gordo.html">`
- **An internal link**: `<a href="#king_kong">`
- **An e-mail link**: `<a href="mailto:primates@famousprimates.com/">`

The first example, an external link, contains the full path to the file we want to link to on the Web. The second example, a local link, contains the path to the file we want to link to *in relation to the file we're linking from*. The third example, an internal link, links to a specific section of the page that the user is *currently on*. The last example, an e-mail link, allows users to contact us via e-mail.

Absolute vs. relative links

Absolute links or relative links, what's the difference? We touched on this previously, but it can be a confusing topic so we're focusing on it here.

Let's imagine Cheeta's neighbor at the Primate Sanctuary, Bonzo, needs to pass Cheeta's address details on to King Kong. Bonzo uses Cheeta's full address, providing everything needed to contact Cheeta: country, state, city, and so on—as in the following example:

Cheeta
The Primate Sanctuary
Palm Springs
CA
USA

This is similar to an **absolute link** and as a URL it might look something like the following:

```
<a href="http://www.usa.com/ca/palm_springs/➥
the_primate_sanctuary/cheeta.html">Cheeta's Address</a>
```


However, because Bonzo lives next door to Cheeta, he can use a shorter address, expressing Cheeta's location in relation to his own. Cheeta is his neighbor after all, so he uses the relationship between his house and Cheeta's, as in the following example:

Next door

This is similar to a **relative link**, and as a URL it might look something like the following:

```
<a href="next_door/cheeta.html">Cheeta's Address</a>
```

Both of the preceding methods point to the same location, so why use one rather than the other? Clearly the relative link is shorter, which is an added bonus, but the primary reason is that relative links become important as we begin to organize the different files that comprise the web site we're creating, gathering them together in logically structured folders, an exercise we embark on in the next section. As we start to link those files together to create our web site, you'll begin to appreciate the importance of using relative links.

Let's return to our friends Cheeta and Bonzo. But instead of being neighbors at the Primate Sanctuary, imagine they're files for the same web site. We could use absolute links every time we needed to link them together, but as you've seen in the preceding example, the absolute link is much, much longer than the relative link. It's much easier to create links between Cheeta and Bonzo using relative links.

But what if King Kong, from Skull Island, needed to get in touch with Cheeta? He certainly doesn't live next door to Cheeta. If King Kong needed to contact Cheeta, he'd need to use an absolute link.

There are certain situations where you need absolute links, and certain situations where you need relative links. As a rule of thumb, *use absolute links when linking to pages on someone else's web site, and use relative links when linking to pages on your own web site.*

The use of relative links will become more apparent as we start to structure our site in the following section.

Structuring your site

By now you should have a number of web pages and image files within your `homework` folder. Things are—dare we say it—starting to get a little messy.

In this section we focus on some methods of organizing your files that will allow you to plan for future growth. In particular we focus on tidying up your web site by organizing its different files into logical folders and adding an `images` folder where you'll gather the relevant images you're using for the site.

Keeping all your HTML files and images in one folder works as long as you have only a few files and one or two images in your site. But what happens if you have ten or twenty images and the number of web pages you've created starts to grow? The folder starts to become cluttered, and it can soon become hard to find what you're looking for. This is where a little organization comes into play.

Organizing your files and folders

Before your web site becomes large and complicated, it makes sense to spend a little time considering how to structure it so that you can organize your files and folders as you add them. Until this point, we've been working with all of our files—HTML pages and images—stored in one folder (or directory) as in Figure 6-3.

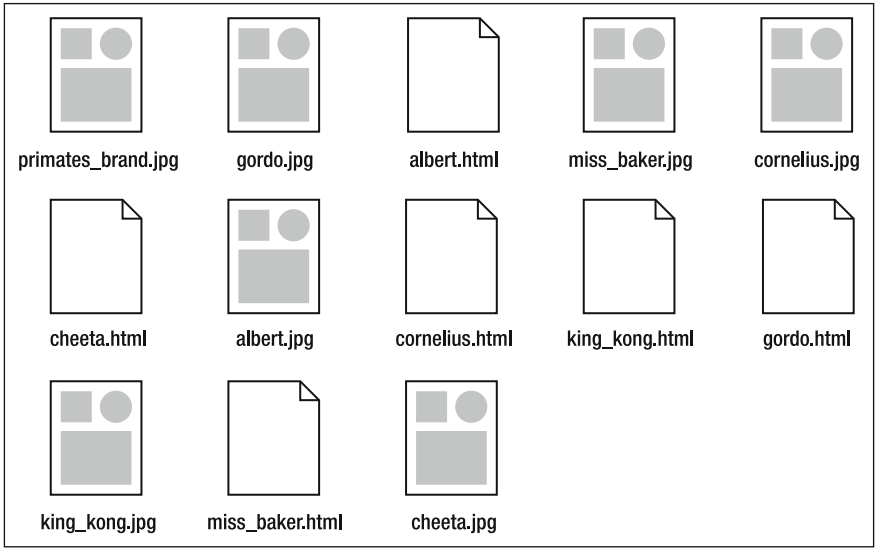

Figure 6-3. Our files as they're currently organized, all in one folder and starting to get a little cluttered

As you can see from Figure 6-3, what we really need now is a little structure to give everything some order. We've left the organization until this point to fit with the structure of the book; however, it's a good idea to consider what your web site might contain and how this information can be logically grouped together **before** you begin developing a web site.

It might surprise you, but at this point in the process, we suggest you turn off your computer and start working with a pen and paper to plan out a site map.

Let's consider what we've created so far. If you've been following along with the homework, you should have created a number of web pages by now. You should have created pages for the following monkey pioneers: Albert I, Miss Baker, and Gordo. Along the way we've provided pages for you to refer to for the following ape thespians: Cheeta, Cornelius, and King Kong. That's quite a few HTML files and images, but we need to add a few more to create a more realistic web site in terms of structure.

With this in mind and to prepare you for this chapter's homework, we'll be providing you with additional files that we have created for you. You'll be organizing and linking these up as part of this chapter's homework.

You'll be aware that there are a number of distinct themes that already suggest some structure, not least the fact that we have two different categories of primates that we could group together. We've covered both monkeys and apes, so we'll create two folders: one where we gather all of our monkey files and a second where we gather all of our ape files.

We could call these folders "monkeys" and "apes," respectively; however, we've also looked at the role of monkeys in the history of space flight and apes in the movie business. So we could give the folders names that reflect that, for example, "pioneers" (for our space-traveling monkeys) and "thespians" (for our apes with acting aspirations).

How we choose to name the folders is important; it has an effect on the character and tone of our web site. Choosing "monkeys" and "apes" as folder titles will result in a web site with a different character and tone than if we choose "pioneers" and "thespians" as folder titles.

Take a look at Figure 6-4, which shows the basic overview of the site we'll be building for the homework with a little structure added. You'll see we've broken the files down into two key sections—pioneers and thespians—which we'll now create folders (or directories) for to keep everything tidy. We've also added a links page, which we'll use to try out some external links for the homework.

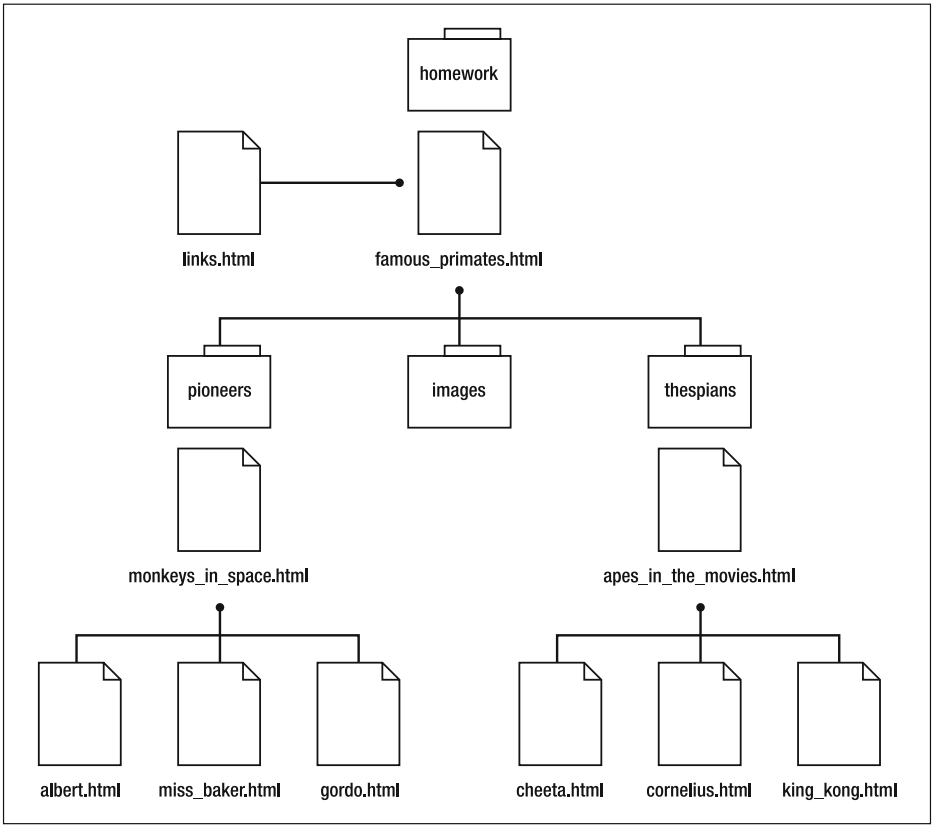

Figure 6-4. Our files as we're going to organize them from now on, using clearly named subfolders

Once you've settled on a site structure and uploaded your web site, resist the urge to restructure your site and change your web pages' locations. Bear in mind that others might have already linked to pages within your site, so changing these files' locations will result in broken incoming links.

One other folder we'd always suggest adding at the start of a project is an `images` folder where we gather images together, specifically using this for any general images employed throughout the site, for example, branding elements.

The magic index file

When you type www.famousprimates.com/pioneers/gordo.html into a web browser's address bar, you should by now know what is going on: the browser will attempt to display a file called `gordo.html` located in the `pioneers` folder. But what happens if you type www.famousprimates.com/pioneers? In this case you don't seem to specify a file for the browser to display; but, as shown in Figure 6-5, a page displays nonetheless.

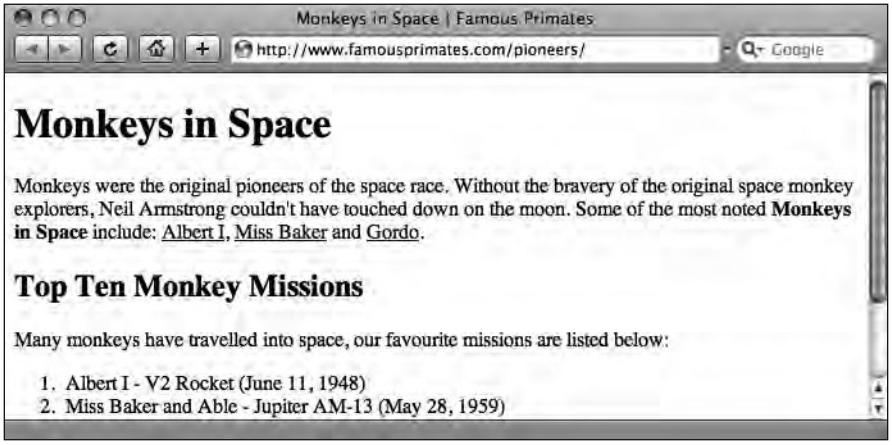

Figure 6-5. Although the URL in the address bar doesn't contain a reference to a specific HTML file, the browser still displays our web page as it will stand after we upload it at the end of Chapter 7.

When you type www.famousprimates.com/pioneers in the address bar, you are telling the browser to look for a folder called `pioneers` at the `www.famousprimates.com` domain. The reason you don't have to expressly add the file name is that web servers are set up by default to look inside the folder and load a specific file known as the **index file**, in our example a file called `index.html`.

> *On some Windows servers the index file is called* `default.html`.

But what happens if you don't have an index file? Depending on your server, you might get an error message when entering a URL that doesn't contain a reference to a specific HTML file, for example, `www.famousprimates.com/pioneers`. Alternatively, and again this depends on how the server is set up, you might get a listing of all the files in the specific folder like in Figure 6-6.

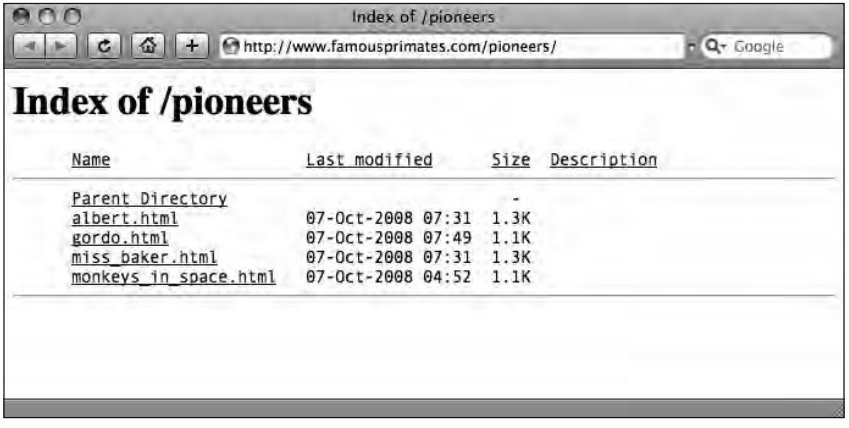

Figure 6-6. Without an index file, we are greeted with the default folder listing. Not pretty.

In preparation for uploading our files to the server on the Web, which we'll cover in the next chapter, we'll make one final amendment to the file structure illustrated in Figure 6-4 earlier. We'll make sure that the files we want to load by default are renamed `index.html`. Figure 6-7 shows our files and folders after renaming `famous_primates.html`, `monkeys_in_space.html`, and `apes_in_the_movies.html` to `index.html`.

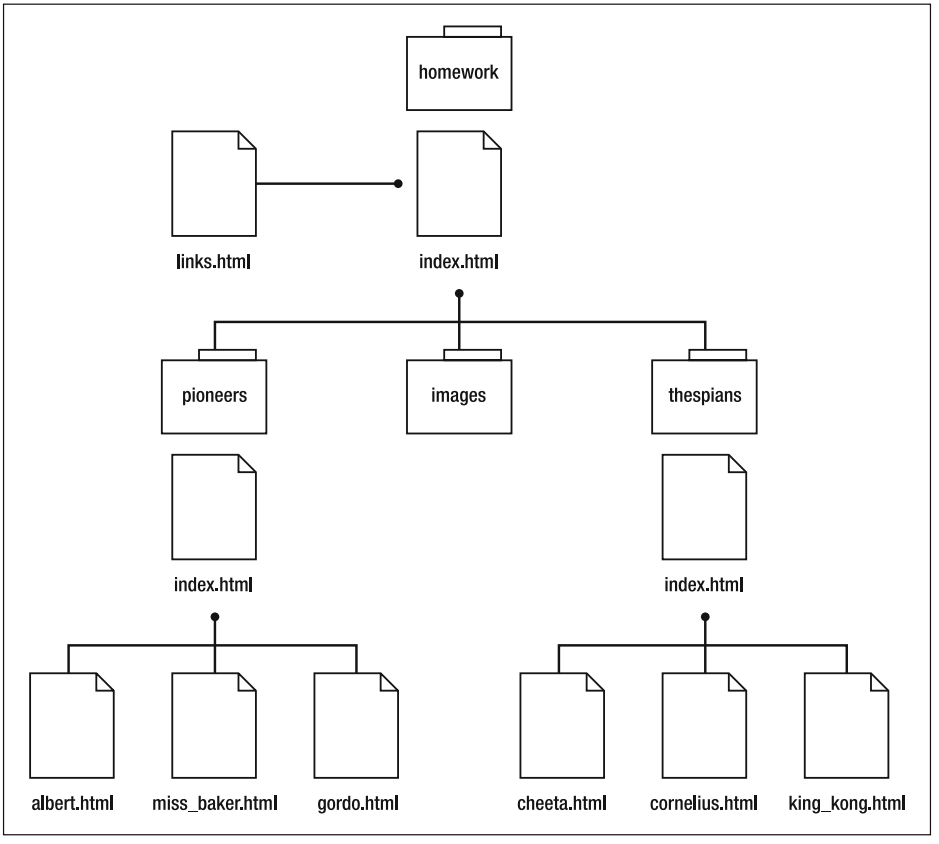

Figure 6-7. Our organized files and folders with the default file in each folder renamed `index.html`

In the next section we take a look at how the different folder titles and, more importantly, how moving our files into these folders will affect our links.

Linking between different folders in our site

Earlier in this chapter we introduced the concept of *relative links*, noting that links between separate files depend upon the relationship of the files' locations to each other. In the last section we moved some of our files around, organizing them into subfolders. Clearly this is going to have an impact on our files' relative locations, which will have changed to take into account that they're now stored in the folders we just created.

This will mean we need to revisit the links we created previously—another reason to think carefully about the structure of your site *before* you begin building it.

Up until this point all of our web pages have been located in a single folder, so linking to them has been as simple as inserting the file name we'd like to link to in our `href` attribute when creating a link. We can still use this method for files stored within the same folder, but we'll need to revisit links between files in different folders to take account of their new relationship to each other. This is best demonstrated with some examples.

Linking within a folder

Take a look at our top-level folder; it now contains only two HTML files, as the rest of the files have now been placed into subfolders. To create a link from the main home page, `index.html`, to the links page, `links.html`, is as simple as follows:

```
<a href="links.html">Links Page</a>
```

This is because these two files sit alongside each other *in the same folder*, as you can see in Figure 6-8.

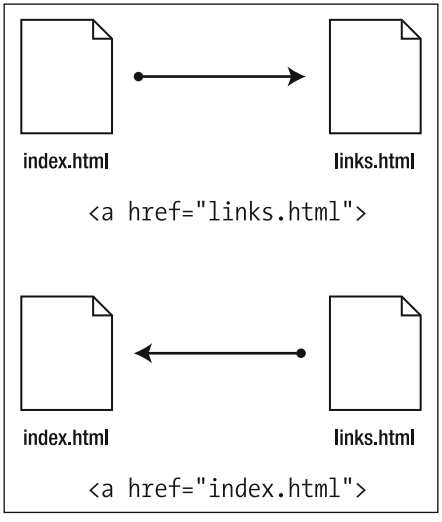

Figure 6-8. Linking between two files in the same folder

Creating a link from the links page back to the home page is equally simple, as shown in the following example:

```
<a href="index.html">Home Page</a>
```

Creating links between files in the same folder is the easiest type of link to create. Let's move to the next stage of complexity, linking from one level of a web site down into a subfolder.

Linking down into a subfolder

Linking down a level, for example, from the main home page, `index.html`, to the `gordo.html` page in the `pioneers` subfolder is slightly different. If you think of the files' relative positions to each other, it should make more sense.

The `gordo.html` page is now located one folder down from the folder containing the `index.html` file, that is, `gordo.html` is located in a *subfolder*. We need to inform the browser of this by giving it the path between the two files as follows:

```
<a href="pioneers/gordo.html">Gordo</a>
```

This is because the file we're linking to is in a folder nested within the main folder as in Figure 6-9. The first part of the preceding `href` attribute, `pioneers/`, tells the browser to look into the `pioneers` folder; the second part, `gordo.html`, tells the browser to look for the file called `gordo.html` inside that folder.

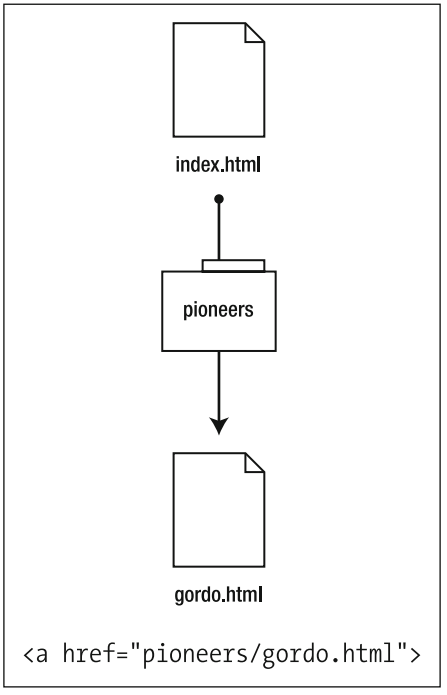

Figure 6-9. Linking to a file in a subfolder

If you've ever typed a typical URL into your browser, you'll have encountered this type of link before, so the structure of the preceding link shouldn't come as a complete surprise.

Linking up into a "parent" folder

For the beginner, linking up into a "parent" folder is probably the hardest concept to grasp, but you should be able to understand the principle if you work through an example. Let's imagine we'd like to create a link from our `gordo.html` page back to the `index.html` home page, that is, in the opposite direction to the link explained in the last section.

The `gordo.html` page is one folder down from the folder containing the `index.html` page. Somehow we need to inform the browser that we'd like to move out of the `pioneers` folder and into the next folder up and look for the `index.html` page.

We inform the browser of the path between the two files as follows (we'll explain it in full in a moment):

```
<a href="../index.html">Home Page</a>
```

The key thing to notice here is the `../` part. This first part of the `href` attribute acts like a sort of magic escalator to move out of the `pioneers` folder and up one level. The second part, `index.html`, tells the browser to look for the file called `index.html` inside the folder we've just moved into, in this case the top-level folder. This is illustrated in Figure 6-10.

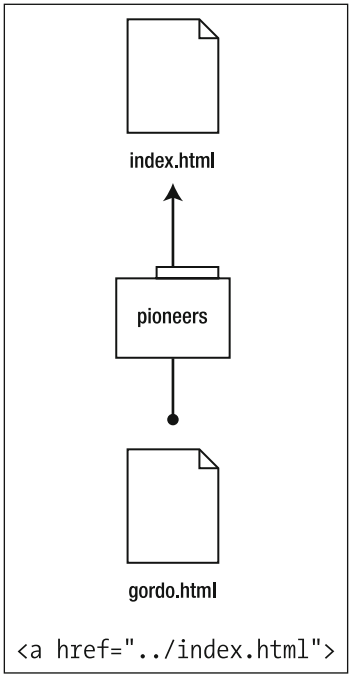

Figure 6-10. Linking to a file in a parent folder, we use the magic escalator `../` to move up one level, out of the `pioneers` folder.

Understanding how to create relative links between files in different folders can be a difficult concept to grasp for the beginner; however, once grasped, the concept will soon fall into place.

> *It's worth noting that you can use multiple instances of the magic escalator, `../`, to move up more than one level in more deeply nested folders. The following example will move up two levels, linking from a file within a folder that is nested within another folder:* `<a href="../../index.html">`.

The best approach to learning this difficult topic is to undertake the exercises in the homework section in which we take you through the relative linking process by moving some files around and asking you to link them together.

Linking up and then linking down

There's just one last thing we need to address now that we've organized all the files . . . our images.

You've seen how to create links between pages in different folders, in particular looking at the escalator—`../`—which lets you link a file in a subfolder to one in its parent folder. Now that we've organized all of our images by placing them in an `images` folder, the relationship of the images and the HTML pages we placed them onto in Chapter 5 has changed.

In Chapter 5 we created our image links as follows (we've omitted the `alt` and other attributes for brevity here):

```
<img src="gordo.jpg" />
```

However, that was when the `gordo.jpg` image was in the same folder as the `gordo.html` page. Now these two different files are in different folders, so their relationship to each other has changed. We need to reflect that by updating our `<img />` tags.

Taking the `gordo.html` page as an example, let's see what we need to do to reestablish the link between the `gordo.html` page and the `gordo.jpg` image. The `gordo.html` page is now contained within a folder called `pioneers`, and the `gordo.jpg` image is now contained within a folder called `images`. Clearly this is going to have an impact on how we write our `<img />` tags so that they reflect the new folder structure.

In order to reestablish the link between the `gordo.html` page and the `gordo.jpg` image, we rewrite the `<img />` tag as follows:

```
<img src="../images/gordo.jpg" />
```

Let's break down what this tag is doing. The first part—`../`—informs the browser to leave the `pioneers` folder where the `gordo.html` file now resides and move up into its parent folder. The second part—`images/`—tells the browser to enter the `images` folder. The final

part—gordo.jpg—tells the browser to display the gordo.jpg image that's now in the images folder. You can see this illustrated in Figure 6-11.

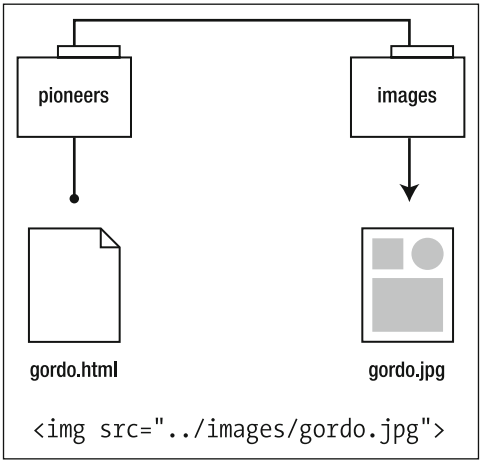

Figure 6-11. Reestablishing the link between gordo.html and gordo.jpg

At first glance, the relationship between links might seem complicated (let's be honest, it is), but practice—as they say—makes perfect. The easiest way to get an understanding of how your different files relate to each other in your newly organized folder structure is to fix the various links that will now be broken as a result of reorganizing your file structure.

In the short term this might seem like a painful process, and the temptation might be to leave everything "organized" in one giant folder. In the longer term, learning about file structure and the relationship between files—in particular the importance of relative links—will stand you in good stead as your web sites grow and expand.

Summary

So what have we covered? This chapter's topic, anchors (or links), is often the biggest obstacle the beginner web designer runs into. Coupling this with the whole topic of organizing files—and the impact that has on links—makes this chapter an important one to grasp.

Understanding relative links isn't easy, and it takes practice through a lot of trial and error. Once the concept is grasped, however, it forms the backbone of the creation of any web site you might build.

After all, a web site wouldn't be a web site without links.

In the next chapter you'll finally take the pages you've created and linked together and upload them to the Web so you can start showing off your work live, worldwide.

Homework: Housekeeping first; links second

In this chapter we've introduced links and their various types: external links, local links, internal links, and e-mail links. We've also introduced the importance of organizing your site by implementing some folder structure to gather your files together in a logical manner.

We looked at how adding a folder structure affected our links, in particular introducing you to the idea of relative links and how they are affected when we restructure our web site and organize its files within folders.

Let's reiterate that point: Relative links are affected as we reorganize our files, so it's a good idea to do the organization *first* and the linking *second*.

This chapter's homework comprises two parts: first, organizing the files you've created so far; second, once that's done, linking up all of the files in your Famous Primates web site.

This is probably the most challenging homework we've set so far and will require some persistence on your part. The topics we've introduced aren't as straightforward as those in the last few chapters, but they *are* important to grasp nonetheless.

1. The content audit

Let's take a look at the files you've created so far. At this point you should have the HTML files `albert.html`, `miss_baker.html`, and `gordo.html` in your `homework` folder. You should also have the images for these files, along with the ape images (that you haven't used yet). You'll be organizing these files as part of this chapter's homework.

2. Here's one we prepared earlier

Let's face it, for many people organization isn't necessarily a task embraced with relish; however, it's important nonetheless. As a little treat for you, and to make your life a little easier, we've provided a ZIP file for you to download at the book's companion web site that already has the correct folder structure created.

This file also contains all of the additional files you need to complete your homework for this chapter, including a page of primate-related information that will act as a home page (this is the `index.html` file in the top-level `homework` folder) and a page of monkey-specific information that will act as the launch page for the monkeys (the `index.html` file in the `pioneers` folder). It also contains our ready-made files for the Thespians section of the site.

You can download the additional files for this chapter's homework from here:

```
www.webstandardistas.com/06/homework.zip
```

Download and unzip this file, and then move the contents, which should look like Figure 6-12, into your `homework` folder.

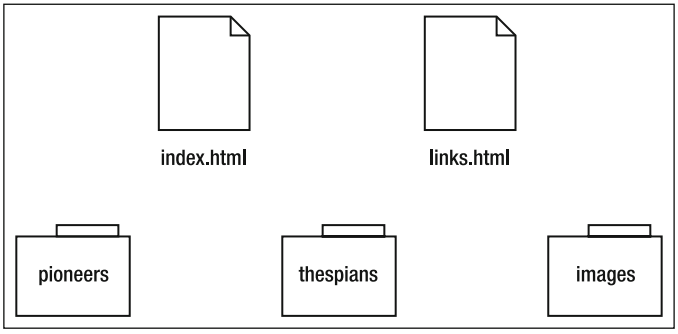

Figure 6-12. The contents of the ZIP file. Move all of this into your `homework` folder.

3. Move your images into the images folder

We left a little bit of organizing for you to do. Now it is time to put all of the images (monkeys, apes, and the brands we have supplied) into the `images` folder. After you've done this, try loading the HTML pages that linked to these images. You should see that the images are no longer showing on your pages; this is because the relationship between the images and the HTML files has changed. You will fix that shortly.

4. Move your monkey pages into the pioneers folder

You now have a folder called `pioneers`. Move your `gordo.html`, `miss_baker.html`, and `albert.html` pages into it. This folder is now the home for all your monkey (pioneer) pages. We have supplied a page of monkey-specific information that will act as the launch page for the monkey pages; this is the `index.html` file already in the `pioneers` folder. That's the organizing over with; now it's time to start looking at our links and images.

5. Fix the image references

As discussed toward the end of the chapter, altering the relationship between the HTML files and the image files they refer to will break the links between them. Now it's time to fix these links. Using the `../` (magic escalator) to move up out of the `pioneers` folder and link down into the `images` folder, amend the image `src` attribute to reestablish links from the `gordo.html`, `miss_baker.html`, and `albert.html` pages to their respective images.

To help you with this you can look at the files in the `thespians` folder to see how these files are linked together. When you've changed the links to reflect the new structure, reload your HTML pages to check that the images display as they should.

6. Add links to the references

Both the Miss Baker and Gordo pages include some references, added in a previous chapter. However, so far these references have not included links. Now is the time to add these. You can get the links for these references in the `monkey_links.txt` file supplied in the `pioneers` folder.

Convert your unlinked ordered lists on the Miss Baker and Gordo pages into lists of links. Again, you can refer to our ape pages in the `thespians` folder to see how this is done.

7. Link your pages together

We've saved the best for last. Now it's time to create internal links from your monkey pages, `albert.html`, `gordo.html`, and `cornelius.html`—each of these pages needs a link to the `index.html` file in the `pioneers` folder. Add a short paragraph to each of these pages that says Back to Pioneers and link it up. You'll see a similar link, Back to Thespians, on our ape pages.

When you're done, check that all your links are working as they should.

This has been a complicated chapter. Once you have all your homework files organized and your links working, you'll certainly have earned the right to put the kettle on and enjoy a cup of *Baker Street Afternoon Blend* as you prepare yourself for the next chapter.