5

Chapter

Useful Web Shell Commands

In this chapter, you create a simple Web page from the ground up and use some of the commands that distinguish the Web shell from a traditional shell. These commands are covered during the course of a tutorial consisting of creating a Web page, retrieving image files from a remote source, and packaging the page and its related files into a single installation file for deployment on any other Web shell system. The Web shell, being a development and deployment environment, makes these aspects of Web development simple.

This chapter begins by walking you through the creation of a Web page modeled on an existing Web page at `www.aestiva.com`, which lists the operating systems on which you can run the Web shell. After this Web page is completed, you learn how to add image files to your Web server across the network using the Web shell's file transfer commands and then add these images to your Web page. When the page is completed, you package all of these files into a single archive file that can be used to redeploy the Web page and all of its images.

Creating a Simple Web Page that Lists Supported Platforms

In this chapter, you create a Web page modeled on an existing Web page on the Aestiva Web site. The Web page will list all of the platforms on which the Web shell can be deployed. There are three steps to completing this Web page. The first step is to create the HTML document. The second step is to retrieve the image files from the Aestiva server and put them on your Web server. The last step is to add the image links to the HTML page and verify your work. After the page is completed, you pack the files you created into an installation file for deployment on other servers that use the Web shell's application packing tool.

Creating the HTML Document

First, you will need to create a directory where you will add your files. Call this directory `platforms` to indicate that the document will relate to the platforms (operating systems) on which the Web shell can run.

To create the directory, run the `mkdir` command with `platforms` as an argument. Remember that `mkdir`, like many shell commands, requires a filename argument with either an absolute or a relative path. If you are in the directory in which the new directory will reside, you need to supply only a relative path, as in the following example:

```
/>mkdir platforms
/>cd platforms
/platforms/>
```

Now you have a directory to hold the data. You put the document and image files in this directory. Use the `edit` command to create the new document:

```
/platforms/>edit index.html
```

When you run this command, the Web shell creates a new edit window in which you will put the contents of the `index.html` file.

Apple MacOS X
[illegible]

Apple MacOS 7-9
[illegible]

Berkeley Systems Design
[illegible]

Compaq Alpha Servers (formerly DEC Alpha)
[illegible]

Hewlett Packard
[illegible]

Figure 5.1 The original Web page at www.aestiva.com.

After issuing the preceding command, you should have an editor window in front of you with a blank text area. You will put the HTML for the `index.html` page in this text area.

The page that you're going to model yours on can be found on the Aestiva Web site. To see the original page, go to `www.aestiva.com` and click any of the operating system icons at the bottom of the page. See the page in Figure 5.1, which shows a part of this page. The Web page you create lists the platforms on which you can run Aestiva, and by extension, the Web shell.

The following is the primary HTML content retrieved from a page posted on the Aestiva site in February 2004. Add it to the text area in the editor window:

```
<html>
<head>
<title>Web Shell — Supported Platforms</title>
</head>
<body>
<h3>Web Shell — Supported Platforms<h3>
<ul>
  <li>Apple MacOS X</li>
  <li>Berkeley Systems Design (BSD)</li>
  <li>Hewlett Packard</li>
  <li>Linux</li>
  <li>Microsoft Windows</li>
  <li>Sun Microsystems — Cobalt Servers</li>
  <li>Sun Microsystems — Solaris-based Servers</li>
</ul>
</body>
</html>
```

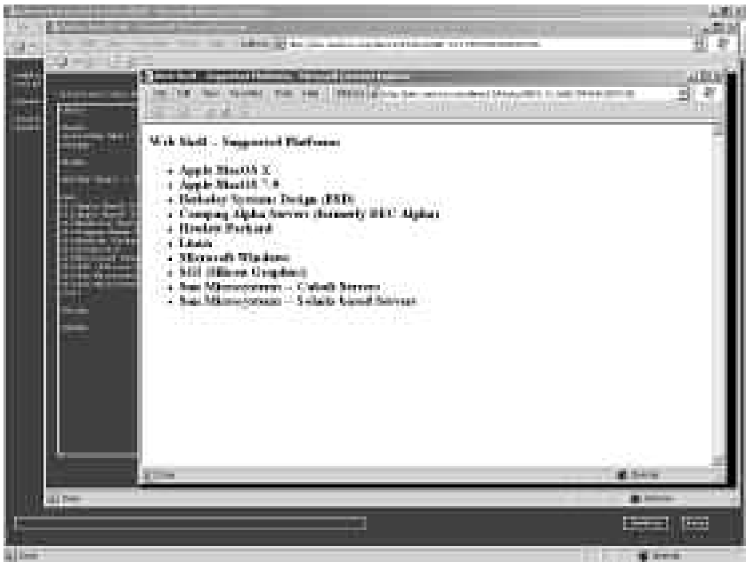
After you finish typing the HTML into the text area, save the file by clicking the Save button. The page reloads and at the top of the editor, the status message should read something like the following, depending on the Web server's time when you saved it:

```
/platforms/index.html FILE SAVED 9:41 AM
```

The next step is to check your work. Click the Run button to launch a new window with your Web page as it will appear when rendered by your Web browser.

When you click the Run button, you should see a newly opened page like that in Figure 5.2. If you see a page like this, you have finished the first step in creating your example page.

The process that you just completed involved writing HTML code and saving it to a server across the network, then viewing the HTML page on that server from your local machine. The process of creating content on a remote server and then testing that content can be



Web Shell – Supported Platforms

- Apple MacOS X
- Apple MacOS 7.8
- Berkeley Systems Design (BSD)
- Compaq Alpha Server (formerly DEC Alpha)
- Hewlett Packard
- Linux
- Microsoft Windows
- SGI (Silicon Graphics)
- Sun Microsystems – Cobalt Servers
- Sun Microsystems – Solaris Intel Servers

Figure 5.2 The platforms Web page lists the supported platforms.

deceptively easy in the Web shell because you don't have to do any explicit networking activity. The network is an integral part of the application environment.

In the next section, you explore some of the more useful aspects of the Web shell's networking features by explicitly transferring image files across the Internet and onto your Web server. You will eventually add these image files to your platforms Web page.

Adding Images to the Web Server

HTML/OS' version of this page at `www.aestiva.com` includes an image next to each operating system. The next step is to add these images to your page. You could easily reference their addresses on Aestiva's Web site directly in the HTML document, thereby causing any visitors to the page to retrieve the image files from Aestiva, but that would be undesirable because you would be “stealing” bandwidth from somebody else’s server, and you wouldn’t have control of the files you are linking to. Instead, you’ll have to add these files to your server and reference them from the HTML document you just created, `index.html`.

You could put the images directly inside of the platform’s directory, but it’s generally neater to put images in their own directory. Create an images directory called `images` inside of the `platforms` directory:

```
/platforms/>mkdir images
```

Now that you have a place to put your image files, you can transfer them from Aestiva’s Web server to your Web server by using the `put` command.

Problems with Accessing Files via FTP

Historically, when developers wanted to move files around on the Web, they used File Transfer Protocol (FTP), which is the most widely used way of transferring files on the Internet. As the name indicates, FTP is not just an application; it’s an entire protocol, distinct and independent of the protocol used by the Web, called Hyper Text Transfer Protocol (HTTP). Part of what that means is that when you want to move a file with FTP, your computer (the file destination computer) has to have an FTP client application running (often confusingly called FTP), and the server (the file source computer) has to have a FTP server running as well. A Web server won’t necessarily have an FTP server installed, and if it is installed, it may not be configured to suit your needs. If you want to get a file from the server, you have to log in with a username and password. After you are logged in, you have to have permission to go into the directory that has the relevant files.

Using the put Command to Upload Web Files

The Web shell uses the `put` command as a substitute for FTP. It won’t always be able to take the place of FTP, because the files you want may be in a directory not served by the remote computer’s Web server. In that case, you can still use FTP to download the remote file to your local computer and use the `put` command from there to upload the downloaded files to the Web server. Otherwise, when it does serve your needs, the Web shell’s `put` command is much more convenient and better integrated than FTP.

In this example, you copy the image files that you will reference from the `index.html` page from the Aestiva's Web server to your Web server. You do this by transferring the files from the source server to the destination server without using your personal computer as an intermediary. You could, on the other hand, download files to your personal computer and upload them, also using the `put` command, but that would require an undue amount of work and network traffic, because you would be effectively transferring files from Aestiva's Web server to your local machine and then transferring the file again to your Web server, as illustrated in Figure 5.3.

Conveniently, the `put` command supports copying files from one Web server to another, without requiring that you use your client as an intermediary. To copy a file from a Web location to your Web server, use the `-u` switch (you can think of the "u" as in URL).

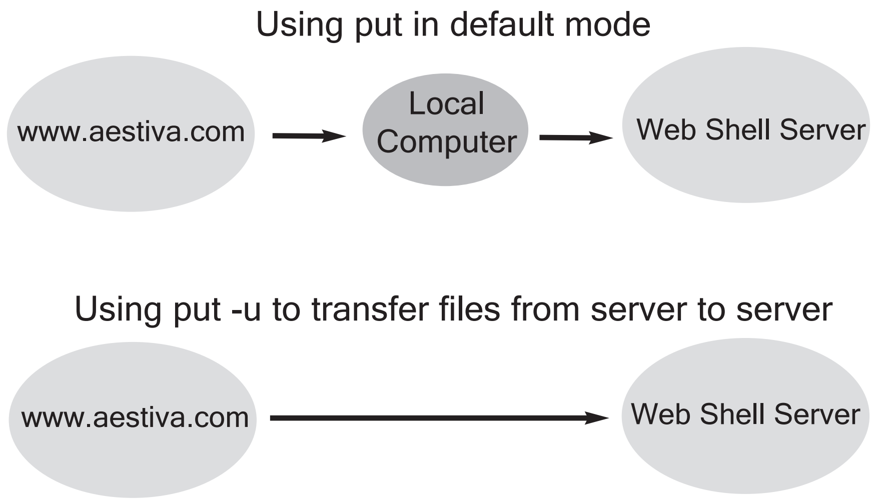

Using put in default mode
www.aestiva.com
Local Computer
Web Shell Server
Using put -u to transfer files from server to server
www.aestiva.com
Web Shell Server

Figure 5.3 The `put` command can either upload files from your personal computer or transfer files from a Web server.

The general form of the `put` command is the following:

```
put [ -u URL] FILENAME
```

When you don't use the `-u` option, the target filename is optional because it is inferred from the source filename. When you do use the `-u` option, the target filename is required.
For example, to copy the image file at:

```
http://www.aestiva.com/aestiva/images/partners/mac_logo.gif
```

to a local file called `/platforms/images/mac_logo.gif`, run the following command:

```
/platforms/images/>put -u
http://www.aestiva.com/aestiva/images/partners/mac_logo.gif
mac_logo.gif
```

This fetches the file located at the specified URL and creates a copy on your Web server named `mac_logo.gif`. See the accompanying note, "The `put` Command and Connection Speed."

The `put` *Command and Connection Speed*

When you run the `put` command with the `-u` option, the Web shell makes an HTTP request from the Web server and writes the contents of the resulting file to the server's file system. Your client computer requests only that the transaction take place; it has no role in the network transaction. That means that if the file were large, and if you were on a computer with a slow Internet connection, the transfer could be fast, depending on the connection between the shell Web server and the source Web server. Typically, the connection between one Web server and another is quite fast, so these transactions are efficient, regardless of the speed of the Web connection of your personal computer.

To view the newly created image file, run the `view` command with the filename as its argument:

```
/platforms/images/>view mac_logo.gif
```

You should see the image appear in a new Web browser window, as shown in Figure 5.4.

Figure 5.4 `mac_logo.gif` in a view window.

Now that you have the Macintosh logo on your Web server, you have to add the rest of the images. Run the following commands to transfer the rest of the files:

```
/platforms/images/>put -u
http://www.aestiva.com/aestiva/images/partners/apple_logo.gif
apple_logo.gif
/platforms/images/>put -u
http://www.aestiva.com/aestiva/images/partners/bsd.gif bsd.gif
/platforms/images/>put -u
http://www.aestiva.com/aestiva/images/partners/digital_logo.gif
digital_logo.gif
/platforms/images/>put -u
http://www.aestiva.com/aestiva/images/partners/ht_logo.gif
ht_logo.gif
/platforms/images/>put -u
http://www.aestiva.com/aestiva/images/partners/linux.gif linux.gif
/platforms/images/>put -u
http://www.aestiva.com/aestiva/images/partners/sun_logo.gif
sun_logo.gif
```

Because typing all of the preceding commands would be tedious and subject to costly typos, this is a good time to use the command history feature of the Web shell. The Web shell remembers your previous commands and lets you recall these commands in several ways. (For more details on accessing the command history, refer to Chapter 3, "The Shell Environment.") The most convenient way to recall a command is by using the up arrow on your keyboard while the cursor is in the command input box. (This feature only works in Internet Explorer, so if you are running another browser, you have to cut and paste your commands into the input box to save typing.)

If you are running IE, press the up arrow to recall the first `put` command that you ran:

```
/platforms/images/>put -u
http://www.aestiva.com/aestiva/images/partners/mac_logo.gif logo.gif
```

Then you edit that command so it refers to the second file in the example, `apple_logo.gif`. The only text values you have to change are `mac` to `apple` in the URL and the filename. You can use the same technique to generate all the remaining `put` commands.

Checking Your Work

When you are done putting all the image files onto the server, view the results by running the `view` command with a wildcard file argument:

```
/platforms/images/>view *
```

Because the `*` character matches all files in the current working directory, the `view` command runs against all of the filenames of the images you just uploaded. You should see a window that displays all the matching images and their filenames, as shown in Figure 5.5.

Adding the Image Tags to Your Web Page

Now that you have the necessary images on the Web server and have demonstrated with the `view` command that they are accessible from within a Web page, you have to add them to

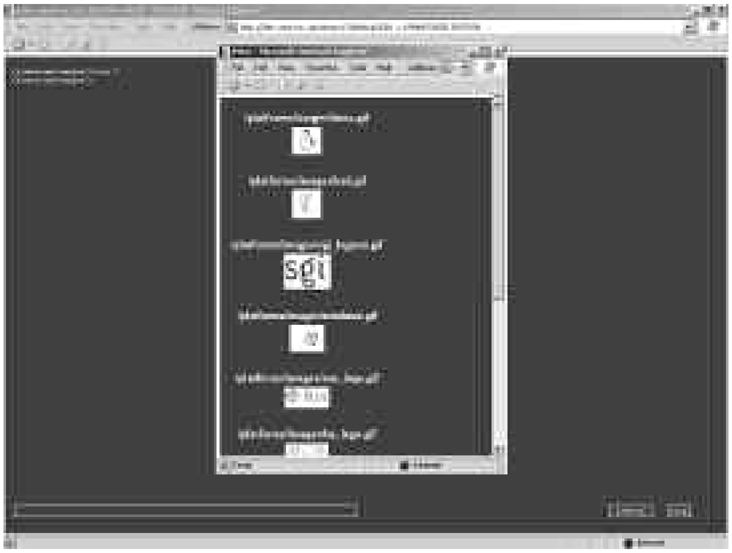


Figure 5.5 Viewing all the platform images.

the platform's Web page. If you don't have the editor window open for `index.html`, open one now with the `edit` command:

```
/platforms/>edit index.html
```

Then add the image tags so that the HTML looks something like this:

```
<html>
```

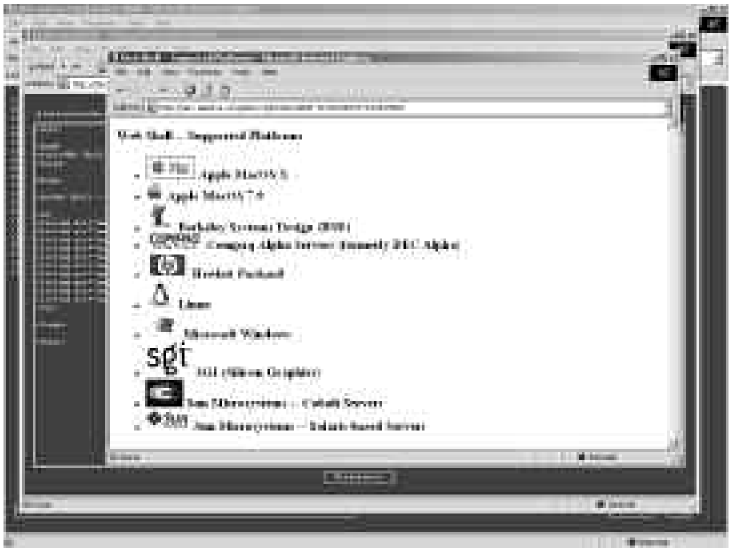
```
<head>
<title>Web Shell — Supported Platforms</title>
</head>
<body>
<h3>Web Shell — Supported Platforms<h3>
<ul>
<li><img src=”images/mac_logo.gif”> Apple MacOS X</li>
<li><img src=”images/apple_logo.gif”> Apple MacOS 7-9</li>
<li><img src=”images/bsd.gif”> Berkeley Systems Design (BSD)</li>
<li><img src=”images/digital_logo.gif”> Compaq Alpha Servers
(formerly DEC Alpha)</li>
<li><img src=”images/hp_logo.gif”> Hewlett Packard</li>
<li><img src=”images/linux.gif”> Linux</li>
<li><img src=”images/windows.gif”> Microsoft Windows</li>
<li><img src=”images/sgi_logonu.gif”> SGI (Silicon Graphics)</li>
<li><img src=”images/cobalt_logo.gif”> Sun Microsystems — Cobalt
Servers</li>
<li><img src=”images/sun_logo.gif”> Sun Microsystems — Solaris-based
Servers</li>
</ul>
</body>
</html>
```

Then run the page to verify your work. Your page should look something like the one in Figure 5.6.

The next step is to package this Web page with its image files into a single archive file that can be deployed on any Web shell system.

Encapsulating and Deploying Web Applications

Often, when you do Web development, you create and verify a Web-based system in a development environment, either on the same server as the deployment environment but in a different directory, or on a different server altogether. In any case, you will often have to move an entire Web application, with source code, HTML documents, code libraries,



Web Shell - Supported Platforms

- Apple MacOS X
- Apple MacOS 7-9
- Berkeley Systems Design (BSD)
- Compaq Alpha Server (formerly DEC Alpha)
- Hewlett Packard
- Linux
- Microsoft Windows
- SGI (Silicon Graphics)
- Sun Microsystems - Cobalt Server
- Sun Microsystems - Solaris-based Server

Figure 5.6 The completed platforms Web page now contains all the image files you inserted.

images, and other files, as well as the directory structure that contains and organizes these files, to the environment that will host the deployed application. In the next section, you package the platforms page and its image files into one file and deploy this mini-application in a different directory on your server.

Creating an Installation File with the pack Command

The `pack` command bundles the files in a Web application, compresses and archives the files into one installation file, and embeds directory structure information so that the application can be unpacked to create a clone of the original.

To pack files, invoke the `pack` command with the name of the archive as its first argument. The following arguments are considered the file sources from which the `pack` command draws to create the archive file.

- To create a pack file, you must specify the `TARGET` pack file. If the file exists, use the `-c` option to overwrite it. The arguments following the `TARGET` file are the files and directories to be packed; any directories given will be recursively packed. Full or relative pathnames are allowed.
- By default the `BASE_DIRECTORY` is the current working directory unless otherwise specified by the `-b` option. All files to be packed must exist somewhere

within the `BASE_DIRECTORY`.

- To specify protected files, use the `-p` option followed by a space-delimited list of the files to be protected. If the list contains more than one file, the entire list must be wrapped in quotes. All pathnames to the protected files must be relative to the given base directory (`cwd` by default).

  To extract (unpack) a pack file, use the `-x` option followed by the pack filename. By default, the pack file will be extracted to the current working directory unless otherwise specified by the `-b` option. To extract only specified files, use the `-p` option followed by a list of files to be extracted. The list of files must be space-delimited and wrapped in quotes. To view the contents of an existing pack file, use the `-l` option.

  Table 5.1 gives a list of the available options.

The `pack` command is one of the more feature-rich commands supported by the Web shell, but for typical use, it is quite simple.

Option	Function
`-c c TARGET`	Creates a pack file and forces the overwriting of `TARGET` if it already exists
`-p "FILE..."`	Protects the specified files (for packing only)
`-b BASE_DIRECTORY`	Set the base directory (`cwd` by default) for extraction or packing
`-n`	Sets no compression (for packing only)
`-l`	Lists files embedded in given pack file
`-x`	Extracts the given pack file `-f "FILE..."` Extracts only the specified files

Table 5.1 Available Options for the `pack` Command.

Packing the Platforms Page and Images

This section shows you how to use the `pack` command to create an installation file that can be used to redeploy the platforms page you just created. You pack up the HTML file as well as all of the images into one file called `platforms.pak`. The following files are associated with the platforms page and are going to make up the contents of the pack archive file:

```
/platforms/>ls *
FILE  PRIVATE             803  05/04/03 15:10:24  index.html
images:
FILE  PUBLIC             1040  05/04/03 23:26:19  apple_logo.gif
FILE  PUBLIC             1690  05/04/03 23:26:23  bsd.gif
FILE  PUBLIC             1520  05/04/03 23:26:38  cobalt_logo.gif
FILE  PUBLIC             1874  05/04/03 23:26:42  digital_logo.gif
FILE  PUBLIC             1194  05/04/03 23:26:59  hp_logo.gif
FILE  PUBLIC             1844  05/04/03 23:26:50  linux.gif
FILE  PUBLIC             1289  05/04/03 23:26:55  mac_logo.gif
FILE  PUBLIC              698  05/04/03 23:27:05  sgi_logonu.gif
FILE  PUBLIC              481  05/04/03 23:27:09  sun_logo.gif
FILE  PUBLIC             1768  05/04/03 23:27:13  windows.gif
```

You could pack these files by specifying them individually as arguments supplied to the `pack` command; but it would be much easier to specify their directory. The following command packs all of the files in the previous list (take note of the current working directory, `/`):

```
/>pack platforms.pak platforms
target file: /platforms.pak
compression: on
archived files:
platforms/images/linux.gif
platforms/images/bsd.gif
platforms/images/sgi_logonu.gif
platforms/images/windows.gif
platforms/images/sun_logo.gif
platforms/images/hp_logo.gif
platforms/images/cobalt_logo.gif
platforms/images/apple_logo.gif
platforms/images/digital_logo.gif
```

```
platforms/images/mac_logo.gif
platforms/index.html
```

The pack command, when used in this way, requires two arguments. The first argument, in the preceding example, platforms.pak, is the name of the archive file you are about to create. The following arguments, in the preceding case one argument, platforms, are the names of the files and directories that you want to include in your archive file. When you run the pack command, it lists the individual files it added to the archive file. In the preceding example, the files added to the platforms.pak archive file are the files required to run the platforms page.

In the previous example, when the pack command runs, you move out of the directory that contains the files to pack and reference the base directory of the Web application from there. This is the simplest method for packing an application and should serve most of your needs. In general, move into the directory immediately above the directory that contains the Web application code (if you consider the root directory to be at the top of the file hierarchy) and give pack the name of that container directory as its only source file argument. The pack command adds all files and folders as deep as the file tree goes from that container directory.

Listing Embedded Pack Files with pack -l

At this point if you were to give platforms.pak to another developer, he or she could examine the contents of the .pak file by running the pack command with the -l option. Run the pack command with the -l option to view the contents of the .pak file you just created, platforms.pak:

```
/>pack -l platforms.pak
platforms.pak:
platforms/images/linux.gif          CPUBLIC     1850
platforms/images/bsd.gif            CPUBLIC     1696
platforms/images/sgi_logonu.gif     CPUBLIC     704
platforms/images/windows.gif        CPUBLIC     1774
platforms/images/sun_logo.gif       CPUBLIC     487
platforms/images/hp_logo.gif        CPUBLIC     1200
platforms/images/cobalt_logo.gif    CPUBLIC     1259
platforms/images/apple_logo.gif     CPUBLIC     695
platforms/images/digital_logo.gif   CPUBLIC     1880
platforms/images/mac_logo.gif       CPUBLIC     1254
platforms/index.html                CPRIVATE    668
```

The resulting list displays the contents of the embedded files. The first column lists the file name, the second column lists the attribute of the file, and the third column lists the uncompressed size of the file.

These details tell you a lot about which files are contained in the `.pak` file, and how the archive will be installed. The first column, for example tells you that when the installation is performed, a `platforms` directory will be created with an `images` directory underneath.

Note that the file paths in the first column are relative to the root directory. In general, when you pack files, they will be assigned a path relative to the base directory at the time of the creation of the pack file. By default, the base directory is the current working directory at the time of creation. You can override this by using the `–b` switch and specifying a base directory other than the current working directory.

The second column tells you whether the files are public or private. (Refer to Chapter 2, "File Management." for more information about private versus public files.) When the archive is extracted, the files listed will be created on the public or private side based on the attributes listed when you run `pack -l`. These are the same attributes the files had when the archive was created. The `C` character before the public/private indication, tells you that the file is compressed in the archive. The `pack` command compresses files by default. Use the `–n` switch to turn compression off.

If, for example, you want the archive to appear like the following version, you specify that the base directory be the `platforms` directory in the following manner:

```
/>pack platforms.pak –b platforms platforms
target file: /platforms.pak
compression: on
archived files:
images/linux.gif
images/bsd.gif
images/sgi_logonu.gif
images/windows.gif
images/sun_logo.gif
images/hp_logo.gif
images/cobalt_logo.gif
images/apple_logo.gif
images/digital_logo.gif
images/mac_logo.gif
index.html
```

In the preceding example, the argument immediately after the `-b` switch overrides the default base directory, the current working directory. It might look funny to see `platforms platforms` on the command line, but in this case, the first `platforms` binds to the `-b` switch and the second `platforms` specifies the source directory. When you check the contents of the pack file, it is evident the paths are relative to `platforms`:

```
/>pack -l platforms.pak
platforms.pak:
images/linux.gif          CPUBLIC    1850
images/bsd.gif            CPUBLIC    1696
images/sgi_logonu.gif     CPUBLIC    704
images/windows.gif        CPUBLIC    1774
images/sun_logo.gif       CPUBLIC    487
images/hp_logo.gif        CPUBLIC    1200
images/cobalt_logo.gif    CPUBLIC    1259
images/apple_logo.gif     CPUBLIC    695
images/digital_logo.gif   CPUBLIC    1880
images/mac_logo.gif       CPUBLIC    1254
index.html                CPRIVATE   668
```

The importance of the paths of the file contents in a pack file arises at installation time. When you unpack a pack file, it gets unpacked into the paths specified by the pack archive. In the preceding case, when you unpack the `platforms.pak` file it will, by default, create the `index.html` file in the current working directory, create an `images` directory relative to the current working directory, and then unpack the remaining image files into that directory.

Unpacking a Pack File

To test unpacking `platforms.pak`, create a new directory called `/deployment` with the `mkdir` command then move into that directory with the `cd` command:

```
/>mkdir deployment
/>cd deployment
```

Run the `pack` command with the `-x` argument to tell `pack` to unpack the file at the given file path, as in the following example:

```
/deployment/>pack -x /platforms.pak
/platforms.pak - OK
```

Keep in mind that the `platforms.pak` file is not in the current working directory anymore because you are in the deployment directory. Use a full path to specify the location of the `platforms.pak` file.

The output of the unpacking process is a status message like the OK message in the preceding example. This tells you whether the unpacking process was successful. You can tell the `pack` command to give you more information by using the `-v` switch to indicate verbose mode. The `pack` command in verbose mode will display the names of the files it unpacked.

Now that the `platforms pack` file has successfully unpacked, list the contents of the current working directory to verify the success of the installation:

```
/deployment/>ls
DIR   MIRROR              360  05/06/03 16:46:41  images
FILE  PRIVATE             803  05/06/03 16:46:41  index.html
/deployment/>
```

Now you can run the `index.html` file with the `run` command.

```
/deployment/>run index.html
```

You should get a Web page exactly like the version you created by hand in the `platforms` directory. Now you have an exact copy of the platforms page as well as all of its images.

If you create an entire Web site with thousands of files, the deployment process is just as simple, provided that the entire application respects the rules of portability. When writing HTML files, the primary rule for portability is referring to embedded content, such as images with relative paths when you write HTML. You should avoid, when possible, using full paths in your `HREF`, `IMG`, and other tags that refer to files on your server. This ensures that a pack file like the one you created in this chapter will be portable across different directories on the same system or on different domain names.

Writing Web Applications for Portability

In the content of the `platforms` page, the image tags referred to files with relative paths, which allowed you to redeploy the page in another directory without any problems. If `index.html` contains references to its image files with full paths, the page references images in the source code area, `/platforms`. Any alteration to the development file system results in an alteration of the production file system.

The following code is an example of a less desirable version of `index.html`. Note that the image files are referenced absolutely:

```
<html>
<head>
<title>Web Shell — Supported Platforms</title>
</head>
<body>
<h3>Web Shell — Supported Platforms<h3>
<ul>
<li><img src="/platforms/images/mac_logo.gif"> Apple MacOS X</li>
<li><img src="/platforms/images/apple_logo.gif"> Apple MacOS 7-9</li>
<li><img src="/platforms/images/bsd.gif"> Berkeley Systems Design
(BSD)</li>
<li><img src="/platforms/images/digital_logo.gif"> Compaq Alpha
Servers (formerly DEC Alpha)</li>
<li><img src="/platforms/images/hp_logo.gif"> Hewlett Packard</li>
<li><img src="/platforms/images/linux.gif"> Linux</li>
<li><img src="/platforms/images/windows.gif"> Microsoft Windows</li>
<li><img src="/platforms/images/sgi_logonu.gif"> SGI (Silicon
Graphics)</li>
<li><img src="/platforms/images/cobalt_logo.gif"> Sun Microsystems —
Cobalt Servers</li>
<li><img src="/platforms/images/sun_logo.gif"> Sun Microsystems —
Solaris-based Servers</li>
</ul>
</body>
</html>
```

If you try to redeploy the preceding file and delete the original `images` directory, the page will not work. In general, keep all paths relative in your application to avoid problems when you deploy the application.

Summary

The functionality outlined in this chapter sets the Web shell apart from other shells, because the Web shell supports the ability to move files across the Web from server to server and create and deploy Web-based applications using a powerful file archiving and unarchiving functionality. You can create, manage, and deploy Web-based applications with relative ease and without having to resort to third-party tools.