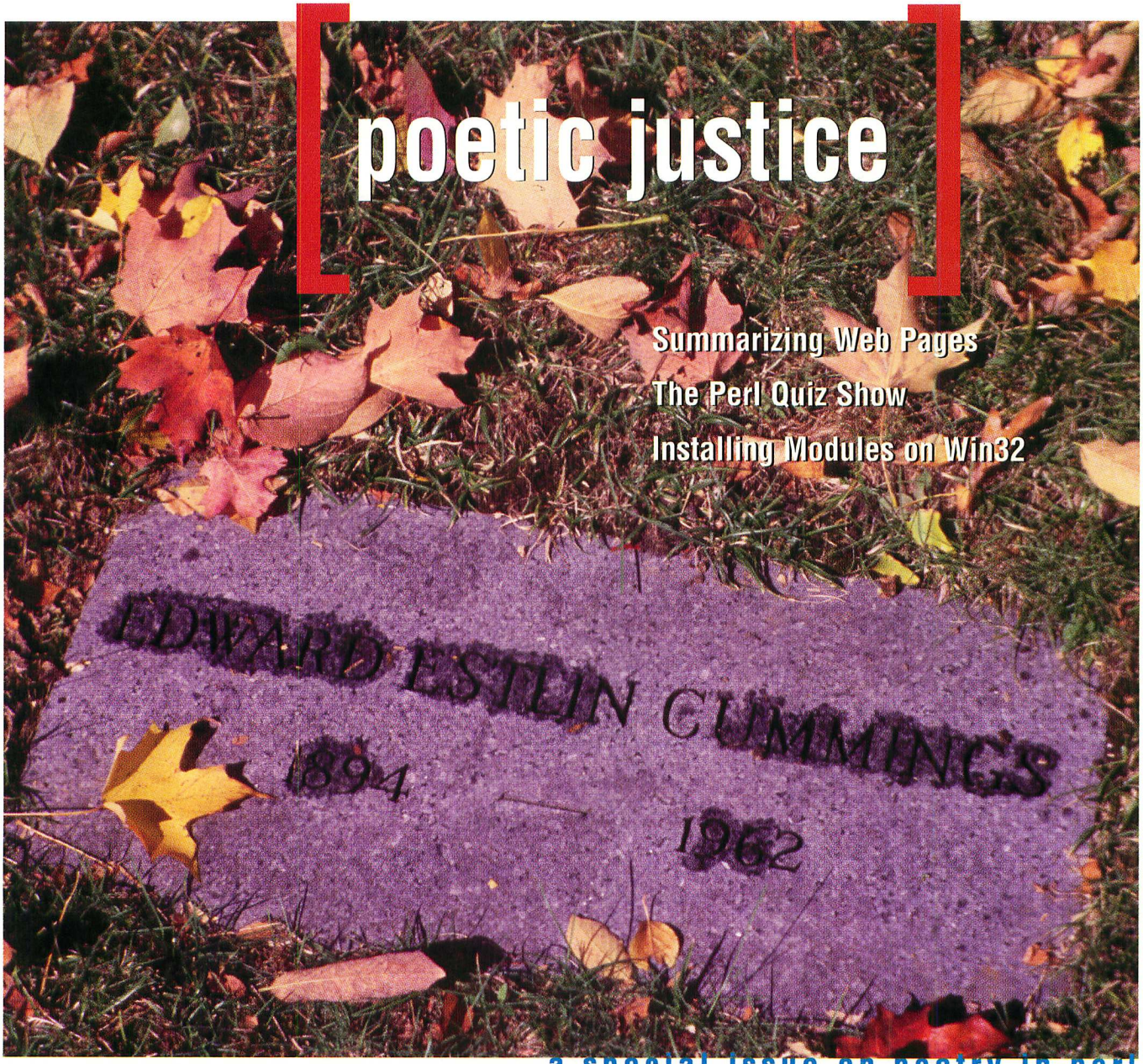
Apache and MP3s • Perl and Java • The Perl Poetry Contest

The Perl Journal

A QUARTERLY MAGAZINE DEVOTED TO THE PERL PROGRAMMING LANGUAGE

ISSUE #16 (VOL. 4, NO. 4) WINTER 1999



poetic justice

Summarizing Web Pages

The Perl Quiz Show

Installing Modules on Win32

a special issue on poetry in perl

$4.95 U.S. Canada $6.50

94

0 74470 90108 2

EARTHWEB®

tpj.com

Recursive Traversal of an FTP Site

Gerard Lanois

Resources

libnet.....http://www.perl.com/CPAN/modules/by-module/Net
libnet FAQ.....http://www.pobox.com/~gbarr/libnet/
RFC 959 .....http://www.yahoo.com/Computers_and_Internet/Standards/RFCs/
Being An FTP Client.....*Perl Cookbook*, O'Reilly, Recipe 18.2
Net::FTP .....*Perl In A Nutshell*, O'Reilly
Win32::Internet FTP Functions .....*Perl In A Nutshell*, O'Reilly

This article is the result of my own personal adventures in maintaining a rapidly growing web site via FTP, without the benefit of a telnet shell on my server. If you have FTP access to your web server's file tree, there are four reasons why mirroring with FTP may be preferrable to HTTP:

1. Your ISP's web server munges links and image paths in your HTML pages, so you can't use HTTP to mirror the site.

2. There is a cache between your HTTP client and your web server, making you retrieve out-of-date pages.

3. Your web site contains dynamically generated content.

4. You have data besides HTML pages and images, such as Perl programs.

This article will demonstrate how to recursively traverse an FTP site using the Net::FTP module bundled with Graham Barr's libnet distribution on the CPAN. For the pedantically inclined, further background information regarding the FTP protocol is available in RFC 959.

MOTIVATION

You may find yourself in the unenviable position of trying to maintain a remote file tree without shell access to the system where your file tree resides. Your file tree might contain a web site, an FTP site, or other data.

Many ISPs do not provide shell accounts, either for security reasons or because the host operating system has no concept of a remote login shell (such as on MacOS or Windows). If you take the login shell out of the equation and wish to automate the process of moving data between file trees on your local machine and your server, a scriptable client becomes a necessity. Fortunately, the Net::FTP module provides an implementation of the FTP protocol so that you can write FTP scripts in your favorite scripting language.

Here are some off-the-shelf approaches to tackling this problem:

1. The classic command-line FTP client.

2. One of the larger fully-featured mirroring tools, such as Lee McLoughlin's `mirror` (http://sunsite.org.uk/packages/mirror/, written entirely in Perl), or Pavuk (http://www.idata.sk/~ondrej/pavuk/).

3. A graphical FTP client, such as `gFTP` (http://gftp.seul.org/), a fairly new but rapidly maturing graphical X Window FTP client based on the gtk+ library, or WS_FTP (http://www.ipswitch.com/), a graphical Windows client.

Each of these tools has its own strengths and weaknesses, and a corresponding place in your toolbox. As my web site has grown over the last couple of years, I have found myself moving individual files and directories with either command-line FTP, or one of the graphical clients mentioned above.

The cornerstone of the Perl philosophy is that "There's always more than one way to do it." I propose the following corollary: "but it's always more fun to do it *your* way". This article will show you how. Here is an amusing anecdote illustrating why I think it's more fun to write your own software:

> An old friend of mine works for one of the big car companies, designing electric cars. One day he described the basic architecture of an electric car, saying "Well, you have some batteries, a motor, a transmission, some software..." I interrupted, "Hold it right there! I write software for a living, and believe me, I don't want ANY software in MY car — at least not any software that I haven't personally written and tested!"

When I stumbled across Net::FTP by accident one day, I began developing a small but effective mirroring program of

my own. I had been avoiding the larger mirroring packages, since I find them to be too (how to say this delicately?) "feature-rich" for my taste.

If you have shell access, mirroring a file tree is trivial:

1. Package up your file tree on your development machine:

```
% cd ~/filetree
% tar cvf - . | gzip > ../filetree.tar.gz
```

2. FTP your package over to the server:

```
% cd ..
% ftp someisp.net
Conected to someisp.net
220 someisp.net FTPServer (Version wu-2.4.2) ready.
Name (someisp.net:gerard): gerard
331 Password required for gerard.
Password:
230 User gerard logged in.
Remote system type is UNIX.
ftp> cd /home/html/users/gerard
250 CWD command successful.
ftp> bin
200 Type set to I.
ftp> put filetree.tar.gz
put filetree.tar.gz
local: filetree.tar.gz remote: filetree.tar.gz
200 PORT command successful. .
150 Opening BINARY mode data connection for
filetree.tar.gz.
226 Transfer complete.
333546 bytes sent in 0.0175 secs (1.9e+04 Kbytes/sec)
ftp> bye
221-You have transferred 333546 bytes in 1 files.
221-Total traffic for this session was 333977 bytes in 1
transfers.
221-Thank you for using the FTP service on lanois.
221 Goodbye.
```

3. Open a shell on the remote server:

```
% telnet someisp.net
Trying 127.0.0.1...
Connected to someisp.net.
Escape character is '^]'.

Red Hat Linux release 6.0 (Hedwig)
Kernel 2.2.5-15 on an i686
login: gerard
Password:
Last login: Mon Oct  4 21:53:57 on tty1
%
```

4. Change directory to the root of the remote file tree (and delete the old file tree, if necessary):

```
% cd /home/html/users/gerard
```

5. Unpack your new file tree:

```
% gunzip < filetree.tar.gz | tar xvf -
```

6. Close the shell on the remote server:

```
% exit
Connection closed by foreign host.
%
```

In the reverse direction:

1. Open a shell on the server:

```
% telnet someisp.net
```

2. Package it up:

```
% cd /home/html/users/gerard
% tar cvf - . | gzip > filetreemirror.tar.gz
```

3. Close the shell on the remote server:

```
% exit
Connection closed by foreign host.
%
```

4. FTP the tree onto your local machine:

```
% cd ~
% mkdir filetreemirror
% ftp someisp.net
...
ftp> get filetreemirror.tar.gz
...
ftp> bye
...
%
```

5. Unpack it on your local machine:

```
% gunzip < filetreemirror.tar.gz | tar xvf -
```

For these two simple cases, an automated Perl client is probably overkill. But take the shell account out of the equation, and you'll find yourself engaging in some *very* long conversations with your FTP server.

NET::FTP

Although the documentation for Net::FTP says that only a subset of RFC 959 is implemented, you will find that the implementation provided by Net::FTP is sufficiently robust for a wide variety of uses. The real power of Net::FTP stems from the power of the Perl programming language itself.

The Net::FTP module is contained in the *libnet* distribution, available from your favorite CPAN mirror in the directory modules/by-module/Net. The filename will be of the form libnet-X.YYYY.tar.gz. As of this writing, the most current version was 1.0607, dated a long time ago: 22-Aug-1998.

There is also a virtually identical FTP capability in the Win32::Internet extension module, although Net::FTP works equally well in both the Unix and Windows environments.

DOWNLOADING A FILE — THE SIMPLE CASE

Here is a short example illustrating how to download a single file; I occasionally use this to download my web server's access log. It is a simple example, but demonstrates all the major steps involved in scripting an FTP session with

Net::FTP.

1. Use the Net::FTP package:

```
use Net::FTP;
```

2. Instantiate an FTP object:

```
$ftp = NET::FTP->new("someisp.net")
  or die "ERROR: Net::FTP->new failed\n";
```

3. Start an FTP session by logging in to the remote FTP server:

```
$ftp->login("anonymous", "g_lanois@yahoo.com")
  or die "ERROR: login failed\n";
```

4. Navigate to the directory containing the file you wish to download:

```
$ftp->cwd("/pub/outgoing/logs")
   or die "ERROR: cwd failed\n";
```

5. Retrieve the file or files of interest:

```
$ftp->get("access_log")
   or die "ERROR: get failed\n";
```

6. End the FTP session:

```
$ftp->quit;
```

RECURSION

Let's quickly review Perl's recursion capability. Recursion barely gets a mention in the perlsub documentation: "Subroutines may be called recursively." This just means that a subroutine can call itself.

Here is a short example which shows how useful this can be. The factorial of a number *n* is the product of all the integers between 1 and *n*. The `factorial()` subroutine below is recursive: it computes the factorial of `$n` as `$n` multiplied by `factorial($n - 1)`.

```
sub factorial {
    my $n = shift;
    return ($n == 1) ? 1 : $n * factorial($n - 1);
}
```

The conceptual model of a file tree is an example of what graph theoreticians call a *directed acyclic graph*. Recursion is the tool of choice when describing algorithms which traverse the nodes of a file tree.

DOWNLOADING A FILE TREE — THE RECURSIVE CASE

On the local machine, if we wanted to crawl a file tree recursively, we would use the `finddepth()` subroutine from the File::Find module. (See Recipes 9.7 and 9.8 in the Perl Cookbook). However, there is no way to perform a `finddepth()` on a remote file tree via the FTP protocol.

Before we tackle the problem of mirroring a remote file tree, let's first develop the technology to crawl the tree. Our approach combines recursion with Net::FTP calls to perform a `find()`-like recursive traversal of the remote tree. Here is a snippet of pseudocode:

```
sub crawl_tree {

    Get a list of all directories and files in the current directory;

    for (each item in the list) {
        if (item is a directory) {
            Save the current FTP remote working directory;
            Change into the directory called "item";
            crawl_tree();
            Restore the remote working directory to what it was before;
        }
    }
}
```

`crawl_ftp` (shown in Listing 1) is a Perl program which traverses a remote file tree, listing the directories and files it finds along the way.

I discovered several interesting issues when developing this script. Any script which uses Net::FTP needs to check for and handle these conditions:

1. `$ftp->cwd` will fail on a directory which has permission set to `d---------`

2. `$ftp->cwd` will succeed on `d--x--x--x`, but `$ftp->dir` will fail on `d--x--x--x`

3. Some (but not all) FTP servers include . and .. when you request a directory listing. Do not recurse on these directories. If you recurse on .., you'll crawl up the tree instead of down. If you recurse on . (the current directory), you'll cause a tear in the space-time continuum, and the computer the script is running on will turn into a Klein bottle.

4. RFC959 does not dictate the format of a directory listing. The following assumptions are reasonable if you don't know the FTP server's listing format in advance:

- The columns in the listing are separated by whitespace.
- The last column contains the file name.
- Directory items in the listing begin with `d`.

5. Handling file names with spaces requires *a priori* knowledge of the listing format. (The `unpack()` function is perfect for parsing out the columns of the directory listing.) The programs in this article do not handle file names with spaces.

Listing 1: crawl_ftp

```
#!/usr/bin/perl -w
# Crawls remote FTP directory.
#     usage: crawl_ftp [-D] host remotedir name password
#         -D  Turn on Net::FTP debug messages

use Getopt::Std;
use Net::FTP;

getopts("D");
defined($opt_D) or $opt_D = 0;

my $host     = shift;
my $dir      = shift;
my $name     = shift;
my $password = shift;

defined($host) and
  defined($dir)  and
  defined($name) and
  defined($password)
            or die "usage: $0 [-D] host remotedir name password";

$ftp = Net::FTP->new($host, Debug => $opt_D ? 1 : 0);

$ftp->login($name, $password);

$ftp->cwd($dir);  # Go to the starting point in the remote tree.

print "DIRECTORY: ",$ftp->pwd,"\n";

crawl_tree(1);    # Crawl over the tree

$ftp->quit;

sub indent {
    # A utility to indent our file tree listing.
    my $num_levels = shift;
    foreach (1..$num_levels) { print "   " }
}

sub crawl_tree {
    my $level = shift;
    my @files = $ftp->dir;  # Make a listing of files and dirs.

    foreach my $i (@files) {
        my @items = split(/\s/, $i);
        my $item = $items[$#items];
        my $parent = $ftp->pwd;

        if ($i =~ /^d(.*)/) {

            # Skip . and .. if present in the listing.
            # (Some FTP servers list these, some don't.)
            next if $item =~ /\.\.?$/;

            indent($level);
            print "DIRECTORY: ", $parent, "/", $item, "\n";

            # Recursively crawl the subtree under this directory.
            $ftp->cwd($item);
            crawl_tree($level+1);

            # Restore location in remote tree.
            $ftp->cwd($parent);
        } elsif ($i =~ /^-(.*)/) {
            # It's a file.
            indent($level);  print "FILE: ", $item, "\n";
        }
    }
}
```

The `crawl_ftp` program shown here produces a nicely-indented listing of the remote file tree.

It would be far more useful to generalize the `crawl_tree()` subroutine, using the same subroutine reference callback mechanism employed by File::Find's `find()` and `finddepth()`. The perlref documentation brushes lightly over the concept of subroutine references, mentioning it in detail only in the context of anonymous subroutines. In our case, it allows us to package our tree crawling technology into a Perl module.

The next listing gives a modified version, with `crawl_tree()` renamed to `ftp_finddepth()` and generalized through the use of a subroutine reference.

Listing 2: crawl_ftp2

```
#!/usr/bin/perl -w
# Crawls remote FTP directory.
#     usage: crawl_ftp [-D] host remotedir name password
#         -D  Turn on Net::FTP debug messages

use Getopt::Std;
use Net::FTP;

getopts("D");
defined($opt_D) or $opt_D = 0;

my $host     = shift;
my $dir      = shift;
my $name     = shift;
my $password = shift;

defined($host) and
  defined($dir)  and
  defined($name) and
  defined($password) or
                  die "usage: $0 [-D] host remotedir name password";

$ftp = Net::FTP->new($host, Debug => $opt_D ? 1 : 0);

$ftp->login($name, $password);

$ftp->cwd($dir);  # Go to the starting point in the remote tree.

print "DIRECTORY: ", $ftp->pwd, "\n";

ftp_finddepth(\&process_item, 1);    # Crawl over the tree

$ftp->quit;

sub indent {
    # A utility to indent our file tree listing.
    my $num_levels = shift;
    foreach (1..$num_levels) { print "   " }
}

sub process_item {
    my ($level, $isdir, $item, $parent) = @_;

    foreach (1..$level) { print "   " }
    if ($isdir) { print "DIRECTORY: ", $parent, "/", $item, "\n" }
    else        { print "FILE: ",$item,"\n" }
}

sub ftp_finddepth {

    my ($callback, $level) = @_;

    my @files = $ftp->dir;  # Make a listing of files and dirs.

    foreach my $i (@files) {
        my @items = split(/\s/, $i);
        my $item = $items[$#items];
        my $parent = $ftp->pwd;

        if ($i =~ /^d(.*)/) {

            # Skip . and .. if present in the listing.
            # (Some FTP servers list these, some don't.)
            next if $item =~ /\.\.?$/;

            # It's a directory - call the callback.
            &$callback($level, 1, $item, $parent);

            # Recursively crawl the subtree under this directory.
            $ftp->cwd($item);
            ftp_finddepth($callback, $level+1);

            # Restore location in remote tree.
            $ftp->cwd($parent);
        } elsif ($i =~ /^-(.*)/) {

            # It's a file - call the callback.
            &$callback($level, 0, $item, $parent);
        }
    }
}
```

The first step is to create a module for the general purpose `ftp_finddepth()` technology we just developed. Then we can write a downloading application that uses the module to traverse the remote file tree's directory structure, transferring any files it finds along the way.

Listing 3: FTPFind.pm

```
package FTPFind;
use strict;

use vars qw($VERSION @ISA @EXPORT);
use Exporter;
$VERSION = 1.00;


sub ftp_finddepth {
    my ($ftp, $callback, $level) = @_;

    # Make a listing of files and/or directories.
    my @files = $ftp->dir or die "ERROR: dir() failed\n";

    foreach my $i (@files) {
        my @items = split(/\s/, $i);
        my $item = $items[$#items];
        my $parent = $ftp->pwd;

        if ($i =~ /^d(.*)/) {

            # Skip . and .. if present in the listing.
            # (Some FTP servers list these, some don't.)
            next if $item =~ /\.\.?$/;

            # It's a directory - call the callback.
            &$callback($level, 1, $item, $parent);

            # Recursively crawl the subtree under this directory.
            $ftp->cwd($item) or die "ERROR: can't cwd() to $item\n";
            ftp_finddepth($ftp, $callback, $level+1);

            # Restore location in remote tree.
            $ftp->cwd($parent) or die "ERROR: can't cwd() to $parent\n";
        }
        elsif ($i =~ /^-(.*)/) {
            # It's a file - call the callback.
            &$callback($level, 0, $item, $parent);
        }
    }
}

1;
```

Writing an application to download a file tree is just a simple matter of writing a `process_item()` callback that mirrors the directory tree and retrieves files, depending on what `ftp_finddepth()` passed it.

If `process_item()` is called with a directory (as indicated by the `$isdir` parameter), we want to create a directory in the local filesystem. If `process_item()` is called with a file, we issue an FTP `get()` request to download the file.

Uploading A File — The Simple Case

Uploading a file is exactly the same as downloading, except you call the Net::FTP `get()` subroutine instead of `put()`.

Uploading A File Tree — The Recursive Case

You would think that using File::Find's `find()` or `finddepth()` would be the way to iterate over the local file tree. There is one small problem with this approach: `find()` and `finddepth()` report the *full* path name of the local directories they find. We only want *relative* local path names of each directory, so that we can duplicate the relative file subtree on the remote system.

We can get by without a remote `mkpath()`-like capability on the

Listing 4: mirror_get

```
#!/usr/bin/perl -w
# Transfers remote file tree to the local machine.
#     usage: mirror_get [-d -D] host remotedir name password
#         -d  Debug mode - don't actually transfer anything.
#         -D  Turn on Net::FTP debug messages

use Getopt::Std;
use Net::FTP;
use FTPFind;
use Cwd;
use File::Path;

getopts("dD");
defined($opt_d) or $opt_d = 0;
defined($opt_D) or $opt_D = 0;

my $host     = shift;
my $dir      = shift;
my $name     = shift;
my $password = shift;

defined($host) and
  defined($dir)  and
  defined($name) and
  defined($password) or
                    die "usage: $0 [-D] host remotedir name password";

$ftp = Net::FTP->new($host, Debug => $opt_D ? 1 : 0)
    or die "ERROR: Net::FTP->new() failed\n";

$ftp->login($name, $password) or die "ERROR: login() failed\n";

# Assume binary transfers.
$ftp->binary;

# Go to the starting point in the remote tree.
$ftp->cwd($dir) or die "ERROR: can't cwd() on $dir\n";

my $root = cwd;    # Remember local root directory.

print "DIRECTORY: ", $ftp->pwd, "\n";

ftp_finddepth($ftp, \&process_item, 1);    # Crawl over the tree

$ftp->quit;

sub process_item {
    my ($level, $isdir, $item, $parent) = @_;

    foreach (1..$level) { print "    " }
    if ($isdir) {
        print "DIRECTORY: ",$parent,"/",$item,"\n";
        if (!$opt_d) {
            # Prepend the remote path with a . and hang it
            # off the directory where we started.
            my $path = "." . $parent . "/" . $item;
            chdir($root)  or die "ERROR: can't chdir() to $root\n";
            mkpath($path) or die "ERROR: can't mkpath() $path\n";
        }
    } else {
        print "FILE: ", $item, "\n";
        if (!$opt_d) {
            chdir($root) or die "ERROR: can't chdir() to $root\n";
            chdir("." . $parent) or
                        die "ERROR: can't chdir() to .$parent\n";
            $ftp->get($item) or die "ERROR: get() failed on $item\n";
        }
    }
}
```

remote system, since we can mirror the local directory to the remote site on the fly as we descend the local tree. We will keep track of our relative location in the local file tree by pushing each directory we descend into onto the back of a Perl array.

So, leaving File::Find's `find()` and `finddepth()` behind, we'll develop our own `finddepth()`. Longtime users of Perl might remember the old example program called `down` distributed with Perl 4. Our version, called `finddepth_gl()` (shown below) performs a similar function — but more portably, since it doesn't involve invoking a Unix command via the Perl `system` function.

Beware that Net::FTP's `mkdir()` will return failure if the directory already exists.

APPLICATIONS

The ability to automate FTP operations relieves a great deal of tedium from having to manually push and pull files to and from your remote file tree. This is particularly useful for periodic and repetitive tasks such as log file retrieval, or unattended updating of an otherwise static web site.

The mirroring applications given above only scratch the surface of what is possible, given a generalized and recursive FTP site traversal mechanism. This gives you the ability to grind over your entire remote file tree. In the case of a web site, this is particularly helpful for rooting out missing or orphaned files. Another application is to automatically check and fix the permissions on all the files in your remote tree. Do you remember the last time you had to do *that* by hand?

_ _END_ _

*Gerard Lanois is a Un*x junkie who has been carrying the monkey on his back since 1986. He invites you to swap stories and folklore with him at* gerard@lanois.com.

Listing 5: mirror_put

```
#!/usr/bin/perl -w
# Transfers local file tree to the remote machine.
#     usage: mirror_put [-d -D] host localdir remotedir name password
#         -d  Debug mode - don't actually transfer anything.
#         -D  Turn on Net::FTP debug messages
# NOTES
#    remotedir must already exist on the remote server

use Getopt::Std;
use Net::FTP;
use Cwd;

getopts("dD");
defined($opt_d) or $opt_d = 0;
defined($opt_D) or $opt_D = 0;

my $host       = shift;
my $localdir   = shift;
my $remotedir  = shift;
my $name       = shift;
my $password   = shift;

defined($host)         and
  defined($localdir)   and
  defined($remotedir)  and
  defined($name)       and
  defined($password)   or
  die "usage: $0 [-D] host localdir remotedir name password";

$ftp = Net::FTP->new($host, Debug => $opt_D ? 1 : 0)
    or die "ERROR: Net::FTP->new() failed\n";
$ftp->login($name, $password) or die "ERROR: login() failed\n";
$ftp->binary;      # Assume binary transfers.

# Go to the starting point in the local tree.
chdir($localdir) or die "ERROR: can't cwd() to $localdir\n";

# Go to the starting point in the remote tree.
$ftp->cwd($remotedir) or die "ERROR: can't cwd() to $remotedir\n";

# Keep track of directory path as separate elements to
# facilitate mirroring of directory paths on remote system.
my @path;

finddepth_gl(\&process_item, 1);     # Crawl over the local tree

$ftp->quit;

sub process_item {
    my ($level, $isdir, $item, $parent) = @_;
    foreach (1..$level) { print "    " }
    if ($isdir) {
        print "DIRECTORY: ",$parent,$item,"\n";
        $ftp->mkdir($parent.$item) or die "ERROR: can't mkdir ",$parent,$item,"\n";
    } else {
        print "FILE: ",$item,"\n";
        my $save_remote = $ftp->pwd;
        $ftp->cwd($parent) or die "ERROR: can't cwd() to $parent\n";
        $ftp->put($item);
        $ftp->cwd($save_remote) or die "ERROR: can't cwd() to $save_remote\n";
    }
}

sub finddepth_gl {
    my ($callback, $level) = @_;

    # Make a listing of files and/or directories.
    my $cwd = cwd;
    opendir(DIR, $cwd) or die "ERROR: can't opendir() on $cwd: $!\n";

    my ($item, @list);
    while (defined($item = readdir(DIR))) { push(@list, $item)}
    closedir(DIR);

    foreach $item (@list) {
        next if $item =~ /\.\.?$/;
        my $parent = "";
        my $i;
        foreach $i (@path) { ($parent .= $i."/" }
        if ( -d $item) {
            # It's a directory — call the callback.
            &$callback($level, 1, $item, $parent);

            # Recursively crawl the subtree under this directory.
            push(@path, $item);
            my $save_local = cwd;
            chdir($item) or die "ERROR: can't chdir() to $item\n";
            finddepth_gl($callback, $level+1);

            # Restore location in local tree.
            chdir($save_local) or die "ERROR: can't chdir() to $save_local\n";
            pop(@path);
        } elsif ( -f $item) {
            # It's a file - call the callback.
            &$callback($level, 0, $item, $parent);
        }
    }
}
```