

Work

by Jeffreys Copeland and Haemer



Simpler is Better.
– Dick Dunn

I have made this letter longer than usual because I lack the time to make it shorter.
– Blaise Pascal

Simplicity

Some weeks, it just doesn't pay to get out of bed. This week, for example.

As part of a testing project, we've decided that we'd like to explore Perl's alternative to Expect.

Expect, by Don Libes, is a Tcl toolkit for dealing with programs that think they have to be interactive. As with all good languages, as far back as FORTRAN IV and SNOBOL, Expect's justified popularity is partly driven by a first-rate book: in this case, Don Libes' *Exploring Expect* (O'Reilly, 1994, ISBN 1-56592-090-2, <http://www.oreilly.com/catalog/expect/>).

"Programs that think they have to be interactive?" Sure. Just try to write a shell script to automate a program that requires logging in to another machine—say, something involving telnet or ftp. You can not do it, because these expect you to enter a password from /dev/tty, *not* from stdin.

Hence, "Expect."

Expect is useful for all sorts of

things. We were once part of a rush project to port a large, UNIX software package to an older and much more primitive mainframe OS. The first thing you miss is make.

Port it? We don't think so. It was far easier to have Expect 1) open up connections to the foreign machine, 2) "type in" the foreign commands to find date-stamps on remote files, 3) parse the times it gets back, and 4) use the times to drive a make on the UNIX system, which, for its part, told the same Expect session what commands to type in on the foreign system to bring things up to date. UNIX make did all the dependency checking and decision-making. Expect, running on the same UNIX box, interrogated the foreign machine and invoked its compilers and linkers.

We like Expect a lot, but its ties to Tcl are a problem for us. We don't use Tcl enough to be comfortable with it; and we especially don't use Tcl enough to debug it well when things go wrong. And don't tell *anyone*, but our programs

don't always work the first time.

We use Perl in lots of our columns because it's compact; we can present significant programs in a fairly compact form, which lets them fit within our word limit.

Expect, on the other hand, has been something that we've had good luck with in the past, especially for testing. Other testers like it, too: Deja Gnu, the Cygnus test harness, is Expect-based.

Long ago and far away, Perl had other modules, `Chat.pl` and `Comm.pl`, which could be used for analogous tasks but were substantially more primitive, so we just stuck with Expect but did not use it as much as we probably should have.

For some time now, though, Roland Giersig has been developing and expanding `Expect.pm`, which provides an Expect-like model. It's both in Perl and is advertised as having the added advantage of being object-oriented.

"Great!" we thought. "Let's try using it, remind ourselves how to use

Expect, and make `Expect.pm` part of our day-to-day arsenal of tools.”

Much easier, it turns out, said than done.

Ow!

First, we had to get it. This led to re-configuring CPAN, because we discovered that moving to IP masquerading had made our old configuration invalid in, it turned out, several ways. We don’t even want to know the details, but we had to; why should you?

CPAN running, we tried to install it onto our handiest local machine. Our handiest local machine wouldn’t install it because it wouldn’t pass the self-tests. After not too much time in the perl debugger, we traced it to a failure of `ptsname()` in `<stdlib.h>`. Our next question turned out to be, “if it’s ‘standard,’ why doesn’t it have a man page?”



Remember what we said about our programs occasionally not working the first time? Open source is where it’s at because other people’s occasionally don’t either.

We went to our nearest Sun machine to look. Someone had put a bad memory card in it and it was down for repair.

Enough already. We found another machine `Expect.pm` would install successfully on and used it.

“Oh good. A tutorial. And it’s eight, full, working programs that illustrate how to use the module. Let’s just run them and figure out how they work.”

They didn’t.

Some of the problems were the examples, some of them were our environment, some of them were sleep deprivation. But wouldn’t you think that a tutorial would have at least one program that worked?

Well, what if we just warm up by converting examples from *Exploring Expect*?

Ah, not all things are done exactly the same way. This is not necessarily bad; the `Expect.pm` alternatives are sometimes cleaner. Simple `prints` to perl `Expect` objects often suffice for communication, for example. A second is that in `Expect`, you set timeouts to infinity by setting them to `-1`, while in `Expect.pm`, you set them to `undef` (which seems more “infinity-like” to us). Or at least, that’s what the documentation says. We tried it, then sat down to fix the bug in `Expect.pm` that kept undefined timeouts from working.

Remember what we said about our programs occasionally not working the first time? Open source is where it’s at because other people’s occasionally don’t either. But this time, tell everyone.

The module is still a work in progress, but it mostly works, and it mostly works well. (The bug we had to fix, earlier, was

a `-w` and use `strict` thing.) You can find out about `Expect.pm` at <http://sourceforge.net/projects/expectperl/>. Similarly, Don Libes’ original can be found at <http://expect.nist.gov/>. (We couldn’t look this up because our DNS server just hung. At least our phones are working—for now—and we’ve got friends we can call to double-check those for us.)

Using It

Once we got through our initial struggles, we found a more interesting challenge.

Consider a simple `Expect` program we wrote to automate anonymous ftp. Used like this:

```
get_file ftp.uu.net:/pub/shells/rc/plan9.ps.Z
```

the command grabs a paper on the plan9 shell, via anonymous ftp.

The code:

```
1  #!/usr/bin/perl -w -s
2  # $Id: get_file,v 1.1 2001/04/08 20:39:12
   jsh Exp $
3  use File::Basename;
4  our $debug;
5  use Expect;
6  @ARGV == 1 or die "usage: $0 ftp-site file\n";
7  my ($host, $path) = split(':', shift);
8  die "usage: $0 host:path" unless defined $path;
9  my ($file, $dirname) = fileparse $path;
10 my $t = 30;
11 my $ftp_prompt = 'ftp> ';
12 if ($debug) {
13     $Expect::Debug=1;
14     $Expect::Exp_Internal=1;
15 }
16 my $ftp = Expect->spawn ("ftp", $host);
17 $ftp->expect($t, "Name") or
18 die "Never got username prompt on $host, " .
19     $ftp->exp_error() . "\n";
20 print $ftp "anonymous\r";
21 $ftp->expect($t, '-re', 'Password:\s*') or
22 die "Never got password prompt on $host, " .
23     $ftp->exp_error() . "\n";
24 print $ftp $ENV{USER} . '@' . 'dnsdomainname'
   . "\r";
25 $ftp->expect($t, '-re', $ftp_prompt) or
26 die "Never got ftp prompt after sending
   username and password, " .
27     $ftp->exp_error() . "\n";
28 print $ftp "cd $dirname\r";
```

```
29 $ftp->expect($t, '-re', $ftp_prompt) or
30 die "Never got ftp prompt after attempting
   to change directories, " .
31 $ftp->exp_error() . "\n";
32
33 print $ftp "get $file\r";
34 $ftp->expect($t, '-re', $ftp_prompt) or
35 die "Never got ftp prompt after attempting
   to get $file, " .
36 $ftp->exp_error() . "\n";
37 print $ftp "quit\r";
38 $ftp->expect($t, '-re', "Goodbye.") or
39 die "Never got 'Goodbye.', " .
40 $ftp->exp_error() . "\n";

41 $ftp->hard_close();
```

Just look at that. Realistically, running `ftp` (or `ncftp`) by hand might be easier. Even without the visual clutter of the object-orientation, it seems far too verbose. For example, we felt we should have been able to do lines 24 through 36 with something more like this:

```
request($ENV{USER} . '@' . 'dnsdomainname');
request "cd $dirname";
request "get $file";
```

“Aha!” we thought to ourselves. The CPAN is littered with Simple modules, from the frequently used `LWP::Simple` to the implausibly named `SGML::Simple::BuilderBuilder`, which isn’t even simple to type. Why shouldn’t there be an `Expect::Simple`, that would let us say what we mean?

Our motto: “Make hard things possible, and simple things Simple.pm.”

Besides, we’d never subclassed someone else’s module and we thought it would be fun.

Hah.

Next month, we will show you what we ended up with, after a series of educational design mistakes.

Other

A few months back, we wrote an article about compression. One of the Linux Certification books we were perusing, in preparation for a month’s worth of volunteer technical work in Romania, mentioned that Linux kernels were typically named either `zimage`, if they’d been compressed with `gzip`, or `bzimage`, if they’d been compressed with `bzip2`. We were ignorant of `bzip2`, so we went on a treasure hunt and wound up with a column.

Wrong again.

In keeping with the rest of life this week (if you’re having trouble making sense of the dates, you’re overlooking publication lag), several folks have already written us to say that `bzimage` files have no connection to `bzip2`. Not that it was a bad column or anything, we just got there through a misunderstanding.

This puts us in mind of the story of lithium treatment for manic depression. In a nutshell, the entire treatment—one of the most specific and effective treatments for any psychiatric condition—was discovered in the 1940s by an Australian doctor, who misinterpreted the behavior of what turned out to be nauseated Guinea pigs. You can read details in Solomon Halbert Snyder’s *Drugs and the Brain* (Freeman, 1986, ISBN 0-71676-017-7).

Our thanks in this matter to Scott McDermott, Sean A. Walberg, and Ronny Haryanto for correcting us early and often. Keep those cards and letters coming, folks.

Finally, Fredrik Viklund wrote to us from Sweden to ask whether we knew an easy way to bookmark directories in the shell, the way you bookmark pages in your browser.

We suggested a pair of shell functions:

```
mark() { eval d_$1=$PWD }
go() { eval cd \"$d_$1 }
```

Given this pair, `mark foo` bookmarks the current directory as bookmark `foo` by setting an environment variable, `d_foo`, to the name of the current working directory. A later `go foo` returns to the marked directory.

Storing the bookmarks in a file, to permit sharing them across sessions, is an easy exercise left to the reader.

For now, however, we’ve had all the fun we can stand, and we’re going home to sleep. Until next month, assuming we wake up by then, happy trails. ✍

Jeffrey Copeland (copeland@alumni.caltech.edu) is currently living in the Pacific Northwest, where he spends his time writing UNIX software in a large development organization and fighting damp rot.

Jeffrey S. Haemer (jsh@usenix.org) works at Minolta-QMS Inc. in Boulder, CO, building laser printer firmware. Before he worked for QMS, he operated his own consulting firm and did a lot of other things, like everyone else in the software industry.

Note: The software from this and past Work columns is available at <http://alumni.caltech.edu/~copeland/work/> or alternatively at <ftp://ftp.cpg.com/pub/Work/>.

ATTENTION READERS

For a convenient way to reach all our advertisers, point your browser to:

<http://swexpert.com/market/>



NOW
ONLINE!