Expect and using Expect with PERL.

EXPECT: A TCL based Toolkit for Automating Interactive Tasks.

Leveraging the task specific capabilities of Expect with the power and flexibility of PERL.

Presented to Philadelphia PERLMongers, September 9, 2002.

Topics

What is Expect?

Expect is a programming language based on the TCL tool-kit designed for automating interactive tasks. Expect is about waiting for something to happen and then doing something in response.

History.

Don Libes, a programmer at NIST had programmed a number of ad hoc solutions to automate interactive processes by the late 1980s. He at that time felt that the functionality of what became expect should be integrated directly into the shell, but didn't want to get involved in re-writing a shell. At a USENIX conference in 1990 Don was priveledged to hear one of John Ousterhout's first presentations about TCL. Strcuk with inspiration, Don went home and eight days later had Expect 1.0 in hand. Don's presentation at the following Usenix conference was an immediate success and both TCL and Expect took off afterwards. In 1995 the O'Reilly Expect book was published, and that is pretty much where Expect stands. It does a specific job, and does it really well, and hasn't changed or needed to be changed in about eight years.

Learning Expect, a Quick Tutorial

If you are already familiar with PERL, EXPECT should be quite easy to learn. Like PERL it is a curly braces language. It is based on TCL and imports many features from TCL, such as its REGEX handling.

The most important three expect commands are 'EXPECT', 'SEND' and 'SPAWN'.

SPAWN creates an external process giving control of STDOUT and STDIN to Expect.

EXPECT tells the program to wait until input matching one or more patterns and then perform the associated action.

SEND tells the program to send something.

```
Spawn telnet myhost.myhost.com
expect "login>" ; send "me\r"
expect "password:" send "1234567\r"
expect "#" ; send "logoff\r"
```

As you can see these three commands are enough to get you up and running with a useful program. You'll notice the \r s at the end of each send, \r is a special sequence meaning send a carriage return.

```
# ryerson script to get current directory listing for
# one of the better slackware mirror sites!

set timeout 12
log_file -noappend ryerson.log
spawn ftp
expect "ftp>" ; send "open ncart.scs.ryerson.ca\r"
# There is a variable text string between name and the colon
expect "Name*:" ; send "anonymous\r"
expect "Password:" ; send "brainbuz@brainbuz.org\r"
expect   {
         "failed" { exit }
         "ftp>" { send "pwd\r" }
         }
expect "ftp>" ; send "cd /pub/slackware/slackware-current/\r"
expect "ftp>" ; send "ls\r"
```

```
expect "ftp>" ; send "cd slackware\r"
expect "ftp>" ; send "ls\r"
expect "ftp>" ; send "exit\r"
```

Ryerson is a script to check one of the more reliable Slackware Mirrors for updates. I'd like to use it to demonstrate a number of features.

```
set timeout 12
```

set tells Expect to set a variable, in this case an environment variable, timeout, to a value of 12. The timeout variable controls the timeout event, which is trappable.

```
log_file -noappend ryerson.log
```

log_file designates and opens a session log so that we can later view the recorded session. The noappend directive means that if the file already exists it will be over-written. If we need to specify a path to the file it is always written Unix style, even on Windows, you must use the forward slash not the backslash.

```
spawn ftp
```

spawn ftp opens our FTP process, under the control of expect.

```
expect "ftp>" ; send "open ncart.scs.ryerson.ca\r"
```

This is our first expect, once FTP is started we want to connect. The send command sends the text, in double quotes to the application.

```
# There is a variable text string between name and the colon
expect "Name*:" ; send "anonymous\r"
```

Because we are uncertain of exactly how the server is going to greet us, the wildcard * is used between the parts of the pattern we are certain of.

```
expect "Password:" ; send "brainbuz@brainbuz.org\r"
expect "ftp>" ; send "pwd\r"
expect "ftp>" ; send "cd /pub/slackware/slackware-current/\r"
expect "ftp>" ; send "ls\r"
expect "ftp>" ; send "cd slackware\r"
expect "ftp>" ; send "ls\r"
expect "ftp>" ; send "exit\r"
```

The rest of the script is straightforward wait for something and send the next command.

Expect does not have data types. Quotes are only required when needed as delimiters, by habit any type of text string will be quoted while an integer won't. Expect is case-sensitive.

A couple of other highly useful expect commands are:

eof	Like timeout, eof, is an event handled by expect and occurs when a spawned process exits.
exit	exits the script, passing an optional parameter.
interact	allows an interactive user to interact with the spawned process. If you have a complex login procedure, you could automate login with expect and then continue an interactive session.
puts	print some string to the terminal.

I said earlier that Expect is a curly braces language. The braces are used to define and set off blocks.

```
expect "foo" { send "bar\r" ; exit "foo bar\r" }
expect   {
         "ficus" { puts "shrub\r" ; exit }
         "rhododendron" { puts "shrub\r" ; exit }
         "president" { puts "Bush!\r" ; exit }
         timeout { puts "Waited too long!\r" ; exit }
         }
```

You'll also see it used.

```
for { set count = 1 } { $count < $someothervar } \
         { incr count +1 } { some block of code to execute }
if { $count > 10 } { some block of code to execute }
```

To a PERL programmer the way the braces are used in for and if statements is confusing. As you may notice when variables are assigned or manipulated by a function such as incr, they are bare, but when their values are called on, they are proceeded with a dollar sign.

Commands terminate with a line ending, unless they are within open braces or a bare backslash is used. Expect will sometimes look for the next line as in expect do something. The expect command looks for either a curly braces block or to the next line. You can even simulate nextline with the semi-colon.

PERL and Expect

First many PERL programmers will say, I can do the same thing in PERL by (some method), the question is which method is easier? As a Network Administrator I mainly automate things like FTP and TELNET with it. Which is why I want to discuss three potentially useful PERL modules Net:FTP, Net:Telnet and Expect.

Expect.PM

The Expect Perl Module is relatively new and provides most of the function of Expect. It is accessed in an object oriented syntax, that if nothing else results in extra typing as you must pass code through the $object_variable->{ 'syntax' }. More important it is a new module and doesn't implement everything. Unless you've personally installed it, it is probably not installed on your system so you'll need to download it if you want to experiment with it.

Net: FTP

This module is pretty standard, reliable and works well. For short FTP transactions I prefer the module over an external expect script.

Telnet

The Telnet module has internal problems with system timer functions, notably on Windows. I mainly work on Windows, the expect and timeout functions of expect are the fulcrum on which automation rests. I have no use for a module which cannot reliably provide them.

Backtick or System

In my work I've developed numerous scripts in PERL that write a script, substituting key values and then execute it using system or backtick, review the output and execute another expect script.

The advantage of backtick is that STDOUT from Expect can be read directly into a variable. However, I've occasionally had problems with backtick that weren't present when using system. My understanding is that backtick and system are implemented differently. With system, however, Expect returns everything to STDOUT, which may or may not be desirable. If it isn't, invoke your command line with redirection, so that STDOUT redirects to a file, which will incidentally be identical to the output of log.

Expect Resources

The Expect Homepage: http://expect.nist.gov/

The (only) Expect Book.
Exploring Expect: A TCL Based Toolkit for Automating Interactive Programs.
1995 O'Reilly. Don Libes.
ISBN 1-56592-090-2
http://www.oreilly.com/catalog/expect/

A five page Tutorial on Expect you can view online:
http://www.raycosoft.com/rayco/support/expect_tutor.html