

```

my @ftree = map { # add to parent's Child list ( $found );
  s/\s+$/; push( @ { $node->{Parent}}->{Children} }, $node
my $depth = 0; # count of indent blocks
$depth++; # start a node (though rather than 0
$parent_stack[ $depth ] = $node;
# create node structure $parent, stack[ $depth ] = $node;

```

Expecting Perl

by Mark Schoonover
mark.schoonover@gmail.com



Imagine having to change the password on hundreds of servers. I could log in to each one individually and hope to get done before I retire, or I could use Perl with Expect.pm to accomplish this very quickly, with less error, by automating it. Anything I can manually do on the command line I can automate with the Expect module.

■ Introduction

Expect was originally a little language written for Tcl to pretend to be someone. From the Tcl website:

Expect is a tool primarily for automating interactive applications such as telnet, ftp, passwd, fsck, rlogin, tip, etc.

For this article, I won't be using Tcl, but Perl with Expect.pm. Perl and Expect.pm make a powerful combination to automate the interaction, data collection and modification of servers, switches, routers, on just about any network device that has a command-line interface and remote access.

■ Getting Started

As an example, imagine me as a new system administrator tasked with inventorying every one of these servers. Of course, my predecessor didn't keep any inventory, and I only have host names and passwords. I have to figure out which distribution and kernel version of each server. Perl with Expect.pm to the rescue!

Expect simulates the human interaction, so it's very important I know the command line interface of my system. As an example, when I use ssh to login to a server, I'm expecting a password prompt (unless I'm using ssh keys). Expect.pm sends commands to the system and expects data to come back. It's best to manually try all the commands I want to automate before developing my Expect.pm-based program. When I do this first, I mostly eliminate command errors when developing my Expect script.

■ Connecting and Logging In

I can connect to my target system in several ways, including various Perl modules such as Net::SSH, Net::SSH::Expect, or Net::Telnet, etc. SSH is installed on my systems, so I use that directly. I construct a command, just like I would if I were doing this myself. I execute that command with the spawn method:

```

use Expect;

my $expect = Expect->new;

my $login =
    "/usr/local/bin/ssh $user@$host";
my $expect = Expect->spawn($login)
    or die
    "Can't login to $host as $user: $!\n"

```

Where \$host and \$user are valid host and user names. If this method call dies, I won't be able to send any commands to the host. To help in debugging my Expect scripts, I enable logging by setting \$Expect::Exp_Interval to a true value:

```

#Enables internal debugging
$expect->exp_Interval( 1 );

```

Additionally, I can redirect the logging output to a file:

```

#Record all communication to logfile.
$expect->log_file("$host.log");

```

The log file will contain all communication between my script and the device. I watch this file to see what's going on with my program. I can use the tail command to watch the file as it's updated:

```

shell> tail example.com.log

```

If this is the first time I'll be logging into a system using SSH, I'll have to deal with the authenticity banner that tells me I'm connecting to a new host:

```

The authenticity of host ' 192.168.0.1
(192.168.0.1)' can't be established
RSA key fingerprint is 84:36:d7:18:09:ec:84
:04:a7:ca:78:49:0d:68:39:b0.
Are you sure you want to continue
connecting (yes/no)?

```

I use the expect method with a regular expression to attempt to match this text:

```

$expect->expect(5, '-re', '(RSA)');

```

```

Install Expect: $prompt> cpan Bundle::Expect Test::Expect

```