Expect

From Wikipedia, the free encyclopedia

Expect is a Unix automation and testing tool, written by Don Libes as an extension to the Tcl scripting language, for interactive applications such as telnet, ftp, passwd, fsck, rlogin, tip, ssh, and others. It uses Unix pseudo terminals to wrap up subprocesses transparently, allowing the automation of arbitrary applications that are accessed over a terminal. With Tk, interactive applications can be wrapped in X11 GUIs.

Expect

Original author(s)	Don Libes
Stable release	5.44.1.15 / March 11, 2010
Written in	Tcl
Operating system	POSIX, Windows
License	Public domain[1]
Website	http://expect.nist.gov/

Contents

- 1 Basics
- 2 Examples
- 3 Usage
- 4 Opinion
  - 4.1 Pros
  - 4.2 Cons
- 5 References
- 6 Further reading
- 7 External links

Basics

Expect has regular expression pattern matching and general program capabilities, allowing simple scripts to intelligently control programs such as telnet, ftp, and ssh, all of which lack a programming language, macros, or any other program mechanism. The result is that Expect scripts provide old tools with significant new power and flexibility.

Examples

A simple example is a script that automates a telnet session:

```
# Assume $remote_server, $my_user_id, $my_password, and $my_command were read in earlier
# in the script.
# Open a telnet session to a remote server, and wait for a username prompt.
spawn telnet $remote_server
expect "username:"
# Send the username, and then wait for a password prompt.
send "$my_user_id\r"
expect "password:"
# Send the password, and then wait for a shell prompt.
send "$my_password\r"
expect "%"
# Send the prebuilt command, and then wait for another shell prompt.
send "$my_command\r"
expect "%"
# Capture the results of the command into a variable. This can be displayed, or written to disk.
set results $expect_out(buffer)
# Exit the telnet session, and wait for a special end-of-file character.
send "exit\r"
expect eof
```

Another example is a script that automates ftp:

```
# Open an ftp session to a remote server, and wait for a username prompt.
spawn ftp $remote_server
expect "username:"
# Send the username, and then wait for a password prompt.
send "$my_user_id\r"
expect "password:"
# Send the password, and then wait for an ftp prompt.
send "$my_password\r"
expect "ftp>"
# Switch to binary mode, and then wait for an ftp prompt.
send "bin\r"
expect "ftp>"
# Turn off prompting.
send "prompt\r"
expect "ftp>"
# Get all the files
send "mget *\r"
expect "ftp>"
# Exit the ftp session, and wait for a special end-of-file character.
send "bye\r"
expect eof
```

Below is an example that automates sftp, with password:

```
#!/usr/local/bin/expect -f #<---insert here your expect program location

# procedure to attempt connecting; result 0 if OK, 1 otherwise
proc connect {passw} {
 expect {
   "Password:" {
       send "$passw\r"
       expect {
          "sftp*" {
               return 0
           }
       }
    }
 }
 # timed out
 return 1
}

#read the input parameters
set user [lindex $argv 0]
set passw [lindex $argv 1]
set host [lindex $argv 2]
set location [lindex $argv 3]
set file1 [lindex $argv 4]
set file2 [lindex $argv 5]

#puts "Argument data:\n";
#puts "user: $user";
#puts "passw: $passw";
#puts "host: $host";
#puts "location: $location";
#puts "file1: $file1";
#puts "file2: $file2";

#check if all were provided
if { $user == "" || $passw == "" || $host == "" || $location == "" || $file1 == "" || $file2 == "" }
  puts "Usage: <user> <passw> <host> <location> <file1 to send> <file2 to send>\n"
  exit 1
}

#sftp to specified host and send the files
spawn sftp $user@$host

set rez [connect $passw]
if { $rez == 0 } {
  send "cd $location\r"
  set timeout -1
  send "put $file2\r"
  send "put $file1\r"
  send "ls -l\r"
  send "quit\r"
  expect eof
  exit 0
}
puts "\nError connecting to server: $host, user: $user and password: $passw!\n"
exit 1
```

Usage

Expect serves as a "glue" to link existing utilities together. The general idea is to try to figure out how to make Expect utilize the system's existing tools rather than figure out how to solve a

problem inside of Expect.

A key usage of Expect involves commercial software products. Many of these products provide some type of command-line interface, but these usually lack the power needed to write scripts. They were built to service the users administering the product, but the company often doesn't spend the resources to fully implement a robust scripting language. An Expect script can spawn a shell, look up environmental variables, perform some Unix commands to retrieve more information, and then enter into the product's command-line interface armed with the necessary information to achieve the user's goal. After looking up information inside the product's command-line interface, the script can make an intelligent decision about what action to take, if any.

Every time an **Expect** operation is completed, the results are stored in a local variable called $expect_out. This allows the script to harvest information to feedback to the user, and it also allows conditional behavior of what to **send** next based on the circumstances.

A common use of Expect is to set up a testing suite, whether it be for programs, utilities or embedded systems. DejaGnu is a testing suite written using Expect for use in testing. It has been used extensively for testing gcc and is very well suited to testing remote targets such as embedded development.

You can automate the generation of an expect script using a tool called 'autoexpect'. This tool observes your actions and generates an expect script using heuristics. Though generated code may be large and somewhat cryptic, you can always tweak the generated script to get the exact code.

Opinion

Pros

Expect can be run at regular intervals through the use of cron to encapsulate system administration tasks. This works because Expect merely uses system administration tools already located on the host computer. No extra tools need to be learned. If the programmer has already learned Tcl, then migrating to Expect is a relatively easy transition. The same programming structures and syntax exist, but with additional features built in.

There is large support in the industry for using Expect for many in-house administration tasks. It is widely used by companies such as Silicon Graphics, IBM, HP, Sun, Xerox, Amdahl, Tektronix, AT&T, ComputerVision and the World Bank to run in-house automated testing for development projects, file transfers, account administration, and network testing.

Expect has been ported to Python, Perl and Java languages in various add-on module projects. Subroutines generally are an interpretation of the original version - with equivalent functionality. Once one understands the concept, one can trivially move to other languages as needed.

Cons

Expect inherits the same syntax convention as Tcl, which may seem unfamiliar if accustomed to other script languages. Compared to languages such as bash, csh, and Perl, Expect has a different twist. It is sometimes challenging to remember when a variable must be prefixed with a "$", and when it must not. There are versions of Expect available for Perl and Python for those familiar with their syntax[2][3].

Another limitation is the difficulty in porting Expect scripts between platforms. For example, an Expect script that was written to use several Unix-based tools, might not be suitable if migrated to a Windows platform. If possible, the programmer must find counterpart command-line applications that provide the same information, and this will probably require changing the send/expect's, which can be a major part of the script. This is not an issue if you load tcl, perl or python on the machines in question, and use those languages' native POSIX interfaces for accessing files, and standard POSIX utilities (telnet, ftp etc.) for remote interaction.

A less obvious argument against Expect is that it can enable sub-optimal solutions. For example, a systems administrator needing to log into multiple servers for automated changes might use Expect with stored passwords, rather than the better solution of ssh agent keys. The ability to automate interactive tools is attractive, but there are frequently other options that can accomplish the same tasks in a more robust manner.

Expect cannot automate GUI based tools. This is generally only a problem on Windows where for many tasks a GUI based interface is the only option. In these situations tools like Autohotkey, AutoIt or Winbatch can be used instead.

References

1. **^** "Expect FAQ: Our company policy requires a license to use Expect. Where can we get a license?" (http://expect.sourceforge.net/FAQ.html#q6) . http://expect.sourceforge.net/FAQ.html#q6.
2. **^** http://www.noah.org/wiki/Pexpect Expect for Python (Pexpect)
3. **^** http://bitbucket.org/geertj/winpexpect/ Windows port of Pexpect

Further reading

- Libes, Don (1995). *Exploring Expect: A Tcl-Based Tool for Automating Interactive Programs*. O'Reilly & Associates, Inc. ISBN 1-56592-090-2.

External links

- Expect on SourceForge (current) (http://sourceforge.net/projects/expect/)
- Official homepage (*very outdated!*) (http://expect.nist.gov/)
- The Tcler's Wiki -- Expect page (http://wiki.tcl.tk/201)
- Perl Expect.pm module (http://sourceforge.net/projects/expectperl)

- Pexpect a Pure Python Expect-like module (http://pexpect.sourceforge.net/pexpect.html)
- Empty - expectlike tool to run command-line interactive programs in UNIX shell-scripts (http://empty.sourceforge.net)
- Expect-lite -- a wrapper for expect, making automation easy, quick, and fun (http://expect-lite.sourceforge.net/)
- Article "Advanced Programming in Expect: A Bulletproof Interface" (http://www.cotse.com/dlf/man/expect/bulletproof1.htm)
- ExpectJ - a Java implementation of the Unix expect utility (http://expectj.sourceforge.net/)

Retrieved from "http://en.wikipedia.org/wiki/Expect"
Categories: Scripting languages | Free software programmed in Tcl

- This page was last modified on 11 March 2011 at 15:42.
- Text is available under the Creative Commons Attribution-ShareAlike License; additional terms may apply. See Terms of Use for details.
Wikipedia® is a registered trademark of the Wikimedia Foundation, Inc., a non-profit organization.