Expect

Greg Baker

gregb@ifost.org.au

What is it?

ı - A language for automating interactive programs

ı - A Tcl extension

ı - Public domain

ı - Named after its most useful command

Versions

- - Version 5.31 works with Tcl 8.2 (22 October 1999)
- - Version 5.31.8 (5 Dec 2000)

Useful Expect examples

- passmass
- kibitz
- timed-read, timed-run
- dislocate

Documentation

- - expect(1) man page
- - "Exploring Expect" by Don Libes (O'Reilly and Associates, ISBN 1-56592-090-2).

Four/fore most important commands

ı - [spawn] start a program

ı - [expect] wait for some input

ı - [send] send something to the program

ı - [interact] allow user interaction through

Automating ftp

```
ftp localhost
Connected to localhost.
220 jayanya.ifost.org.au FTP server ready.
Name (localhost:gregb):  user1
331 Password required for user1.
Password:
230- jayanya ppc
230 User user1 logged in.
Remote system type is UNIX.
Using binary mode to transfer files.
ftp> dir
```

The interactions necessary are:

- To put in a username
- To put in a password
- To run some commands

Example Prog

```
#!/usr/bin/expect
spawn ftp localhost
expect "Name" ; send "user1\n"
expect "Password:"
send "class1\n"
expect "ftp>”
send "put myfile myfile.bak\n"
expect "* bytes sent in *ftp>”
send "bye\n"
```

Tcl/Expect Syntax

- - normally one-per-line
- - (i.e. terminated by a new-line character)
- - Can be separated with semi-colons

Problems

- What if we can't reach the FTP server?
- What if the password is wrong?
- What if the connection times out?
- It's a bit talkative

Multi-pattern expect

```
expect pattern action pattern action ...
```

- Use { ... } or "..." for actions or patterns with spaces
- If the input buffer matches a pattern, the paired action is run

The three keywords for patterns

- [eof] To be run on end-of-file
  (e.g. the spawned program completes)
- [timeout] Wait N seconds for input, then try this (N = the "-timeout" parameter or "timeout" variable)
- [default] Either of the two above scenarios.

Informing the user

- send_user "Working!\n" (standard output)
- send_error "Failed.\n" (standard error)
- send_tty "Message\n" (controlling terminal)

Tcl variables

ı - set username "user1"

ı - send "$username\n"

Other things

- [log_user 0]

  Don't print the session transcript
- [expect -i]

  Look for the pattern case-insensitively
- [exit]

  Exit program with appropriate error code

```
set username "user1” ; set password "class1” ; log_user 0
spawn ftp localhost
set timeout 45
expect "220*Name" { send_user "Connected!\n" } \
 timeout { send_error "Never got name\n" ; exit}
send "$username\n"
expect -i "Password"
send "$password\n"
expect "530 Login incorrect" {
     send_error "Bad password\n" ; exit 2 } \
 "230*logged in*ftp>" { send_user "Password OK\n" } \
    timeout { send_error "Login timed out" ; exit 3 }

set timeout 200  ; # could take a while to send
send "put myfile myfile.bak\n"
expect "226 Transfer complete*ftp>" {
    send_user "Transfer OK\n" } \
    timeout { send_error "Transfer not done in time" }
send "bye\n"
```

Exercise Break

1. Automate a telnet connection login
2. Use it to get a directory listing
3. Transfer it to transfer a file
   (hint: cat file ; echo "ZZZ-EASY-TO-RECOGNISE")
4. (Optional) Test whether an email address is valid

Tcl/Expect Syntax

- - everything is a string
- - commands break into "words" (like the Unix shell)
- - five special characters:
- {...} "..." [...] $ \

{ .... }

- acts like Unix shell single-quote (')
- can be nested
- can spread over several lines (common)

" .... "

- " .... " acts like Unix shell double-quotes
- $var gets replaced with contents of "var"
- usual string chars \n \t \a \r
- can spread over several lines

[....]

- a bit like Unix shell backquotes
- run the command between [and]
- command output is a command argument

Stylistic improvements

```
expect {
    "220*Name" {
        send_user "Connected!\n"
     }
    timeout {
        send_error "Never got name\n" ; exit 1
    }
}
```

What commands are there?

- looping, lists, hash/arrays
- procedures, functions
- `info commands`
- commands themselves may give some usage information
- tries `auto_load`

User interaction

- expect_user
- interpreter
- interact

expect_user

- same as normal expect
- reads from user rather than process

interpreter

- starts an expect interpreter
- useful for debugging
- very useful with interact...

interact

```
spawn ftp localhost
expect "Name" ; send "user1\n"
expect "Password:"
send "class1\n"
expect ftp>
interact {
    rm { send delete }
    ~c { send "mget *.c" }
    ~p { send "mget *.pl" }
}
```

multiple sessions

- `spawn` sets the variable "`spawn_id`"
- save `spawn_id` and spawn again
- use `expect -i` and `send -i`

```
spawn telnet host1 ; set session1 $spawn_id ;
interact "^-" return ;# manual login
spawn telnet host2 ; set session2 $spawn_id
interact "^-" return ;# manual login
set timeout -1 ;# no timeout
while (1) {
 send_user "\nid, ls, date or q> "
 expect {
  -i $user_spawn_id
   "id\n" { send -i $session1 "id\n" ;
            send -i $session2 "id\n" }
   "ls\n" { send -i $session1 "ls\n" ;
            send -i $session2 "ls\n" }
   "date\n" { send -i $session1 "date\n" ;
              send -i $session2 "date\n" }
   "q" { break }
   "\n" { }
  -i $session1 -re ".+" { send_user "\n**1>
  $expect_out(buffer)" }
  -i $session2 -re ".+" { send_user "\n**2>
  $expect_out(buffer)" }
 }
}
```

Exercise time

1. Improve the FTP example from - the user should interact to fix a wrong password
2. Extend FTP to copy from sys A to sys B via your machine
3. Fix double-headed-telnet to:

   a) not show the shell prompt

   b) interact again with a session

Walk through of Tcl examples

- sampletcl-basic.exp
- sampletcl-advanced.exp

Modular programming

- source filename
- namespace

Exercise

1 Remove your most time-wasting task