— Expect —
A Tool For Automating Interactive Programs

Don Libes
libes@nist.gov

National Institute of Standards and Technology

Expect – Why?

- A tool for automating interactive programs
    - tip, telnet, ftp, passwd, su, rogue . . .

```
% telnet medlib.iaims.georgetown.edu
Trying 141.161.42.1 ...
Connected to medlib.iaims.georgetown.edu.
Escape character is ‘^]’.
UNIX(r) System V Release 4.0 (medlib)
login: medlib
Enter Your Last Name: mulroney
Password:XXXXXXXX
         ... menu ...
Enter a number or Q to QUIT: 4
```

Talk Overview

- Motivation
- Basics, Examples
- Patterns and Actions
- Autoexpect
- Total automation vs partial automation
- Converting line-oriented apps to window-apps
- Background: CGI, cron, etc
- Security
- Esoterica
- URLs and other pointers

Expect Commands Are Simple

- Start an interactive process

```
spawn ftp tribble.cme.nist.gov
```

- Send a string

```
send "dir\r"
send "get $file\r"
```

- Wait for a response

```
expect "Password:"
expect "200*ftp>"
```

- Pass control from script to user

```
interact
```

Example – /bin/passwd

```
spawn /bin/passwd $user
expect "Password:"
send "$password\r"
expect "Password:"
send "$password\r"
expect eof
```

Another Example: Dial a Modem

- Connecting to a remote site through a modem

```
spawn tip modem
expect "connected"
send "ATZ\r"
expect "OK"
send "ATD1234567\r"
expect "CONNECT"
send "\r\r"
expect "login:"
send "library\r"
interact
```

Tcl – A Scripting Language

- Tcl: a shell-like extensible interpreter by Ousterhout, Winter 1990 USENIX. Example:

```
set temperature 20

if {$temperature < 50} {
     puts "It’s pretty cold."
} elseif {$temperature > 70} {
     puts "It’s really hot."
} else {
     puts "It feels like spring!"
}
```

A While Loop

```
while {$temperature < 60} {
     puts "$temperature is still chilly!"
     heat 10 min
}
puts "Ah, that’s warm enough."
```

- break: break out of a while loop
- continue: continue a while loop from the beginning
- return: return from this procedure

More Password Automation

- Changing passwords on accounts on multiple machines

```
foreach host $hostlist {
    spawn rlogin $host
    expect "$prompt"
    send "/bin/passwd\r"
    expect "Old password:"
    send "$oldpass\r"
    . . .
}
```

- Actual script is parameterized

Passmass – Already Written!

- Handles different access methods (telnet, rlogin, etc.)
- Handles different password programs (passwd, yppasswd)
- Handles different user names, prompts, host equivalencing
- No special knowledge of daemons, password formats, encryption

In The Same Way . . .

- Expect easily controls:
  - VMS systems
  - Printers
  - Modems
  - Pagers
  - Routers
  - Servers
  - Black boxes
  - ...and more
- Expect is also useful for testing
  - Software
  - Hardware
  - Test Suites: Cygnus (DejaGnu), X/Open, VSC4 (test suite for XPG), NIST

Actions

- Tcl’s if command has an action

```
if {$temp == 100} {puts "It’s really hot!"}
if {$temp == 100} {
     puts "It’s really hot!"
}
```

- Expect’s action work the same way

```
expect "100" {puts "It’s really hot!}
expect "100" {
     puts "It’s really hot!
}
```

Pattern/Action

```
expect "pattern" action "pattern" action

expect {
     "pattern1" action1
     "pattern2" action2
     "pattern3" {
         action3a
         action3b
         action3c
     }
}
```

Waiting For Different Responses

```
expect {
     "does not exist" exit
     "password: " {
         send "$password\r"
         # more stuff can done here
     }
}
```

- Actions can include expect commands
- Simple actions do not need braces, example: exit

Waiting For Different Responses – part 2

- Host equivalencing produces different prompts

```
expect {
        "$shellprompt" {
            send "/bin/passwd\r"
        }
        "Password:" {
            send "$password\r"
            exp_continue
        }
```

More On Patterns

- Prompts can include variables

```
expect "$shellprompt"
```

- Glob patterns (Shell-style)

```
expect -gl "catch a falling *"
```

- Regular expressions

```
expect -re "(login|Username):"
```

- Exact strings

```
expect -ex "catch a falling *"
```

Mixed patterns

```
expect {
     -re "3.*ftp>"      action3
     -re "2.*ftp>"      action2
     -gl "ftp>"         actionDefault
}
```

- Expect’s internal pattern matching strategy is intuitive
  - Loop until match
  - Match patterns in order
  - Idle while waiting for more input
- Expect is event-loop compliant

Anchors

- ^ matches the beginning of the buffer
- $ matches the end of the buffer
- Valid for -gl and -re
- Not for -ex

Keywords

```
expect eof action
expect timeout action
```

```
set timeout 60
set timeout -1   ;# no timeout
```

- These are implicit in every expect command. Consider:

```
expect "foo"
```

Timeout Example

- Host equivalencing produces different prompts

```
expect {
        "$shellprompt" {
                send "/bin/passwd\r"
        }
        "Password:" {
                send "$password\r"
                exp_continue
        }
        timeout {
                puts "timed out!"
                continue
        }
```

Alternatives To Expect

- Automating passwd – the hacker approach
  - Get source (if possible) and modify command-line argument handling.
  - If no source...
  - Encrypt passwords
  - Lock/read/write password database
  - God forbid any of these change
  - NIS
  - Kerberos
  - Shadow passwords
  - Rechange, retest, redebug...
- Expect – the solution for the rest of us
- Re-usable on passwd, telnet, others

Partial Automation

- Sometimes, it is inappropriate to totally automate

```
spawn telnet $host
expect "login:"    {send "$name\r"}
expect "Password:" {send "$password\r"}
expect "$prompt"   {send "cd $dir\r"}
interact
```

- interact works in both directions

Example: fsck, The File System Checker

- fsck: a typical vital program with a poor interface
  - fsck -y   or fsck -n   (that’s it for programmability!)

```
while {1} {
   expect {
       eof                          break
       -re "UNREF FILE.*CLEAR.*?" {send"y\r"}
       -re "BAD INODE.*FIX.*?"   {send "n\r"}
       -re "? "                  {interact}
   }
}
```

Interact patterns/actions

```
interact pattern action pattern action . . .
```

```
while {1} {
   expect {
       eof                          break
       -re "UNREF FILE.*CLEAR.*?" {send"y\r"}
       -re "BAD INODE.*FIX.*?"   {send "n\r"}
       -re "? "                     {
           interact "+" return
       }
   }
}
```

Example: Adding Commands To ftp

- Better than fsck in terms of programmability
  - But not much! No reliability.

```
interact {
     "~g\r" {get_current_directory}
     "~p\r" {put_current_directory}
     "~l\r" {list_current_directory}
}
```

list_current_directory

```
proc list_current_directory {} {
     send "dir\r"
     # expect commands to read directory

     foreach file $list {
         if {$isdirectory} {
             send "cd $file\r"
             list_current_directory
             send "cd ..\r"
         }
     }
}
```

Feedback

- expect_out contains results of a match
- expect_out(buffer) contains entire match plus things skipped
- expect_out(0,string) contains entire match
- expect_out(1,string) contains submatch 1
- expect_out(2,string) contains submatch 2
- expect_out(3,string) . . .
- and so on

```
expect -re "a*((ab)*|b*)"
```

Feedback example

```
expect {
    "ld password:" {
        send "$oldpass\r"
    } "assword*:" {
        send "$newpass\r"
    } -re "(.*)\n" {
        showerr "$expect_out(1,string)"
    } eof {
        showerr "passwd died unexpectedly"
}
```

Lots of Other Features

- Global patterns
  - expect_before
  - expect_after
- Multiple processes
  - Example: testing two programs
  - Example: program1 -> program2 -> program1

Tk — An X11 Extension To Tcl

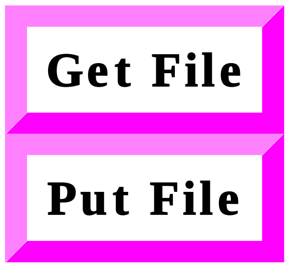
- Tk commands are simple
- Buttons

```
button $gbut -text "Get File" -command get
button $pbut -text "Put File" -command put
```

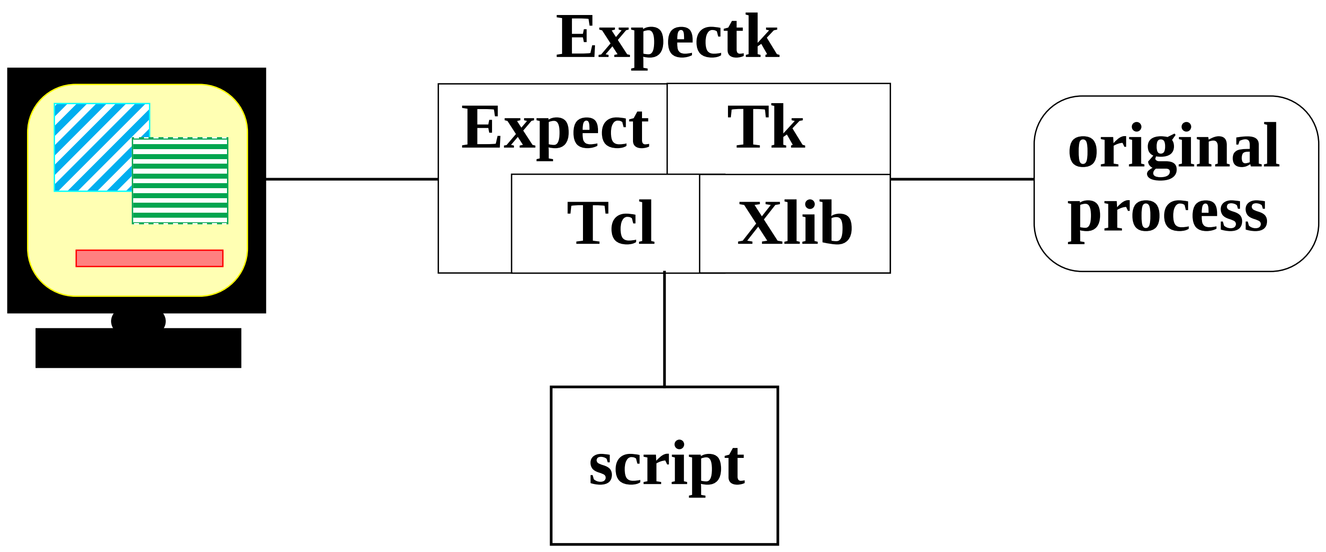
- Bindings

```
bind $window <Button3> swap-directory
```

- Lots of other widgets, Motif-style
  - Window
  - Scrollbar
  - Radio button
  - Check button
  - Canvas
  - Etc

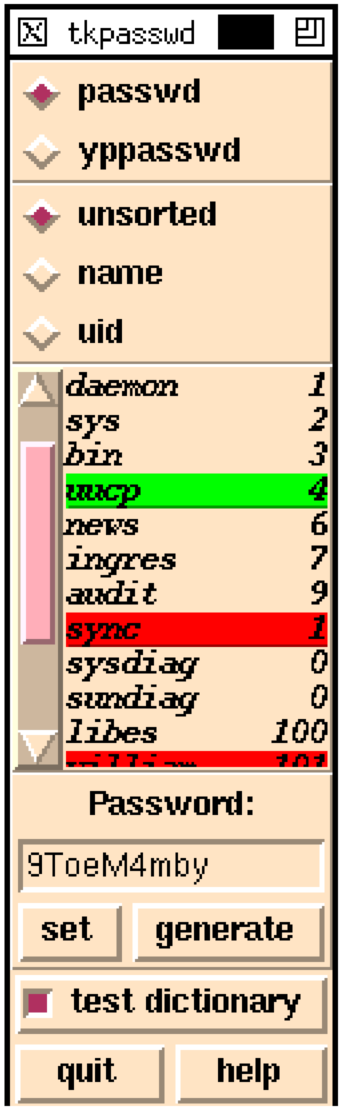


Expectk – Script Driven



- Add scrollbars, buttons, etc. to existing programs
- Or completely cover them up.
- No changes are required to original programs.
    - Ergo, no testing of changes is necessary.

Example – tkpasswd – A Tk GUI for passwd



[✓ passwd] yppasswd

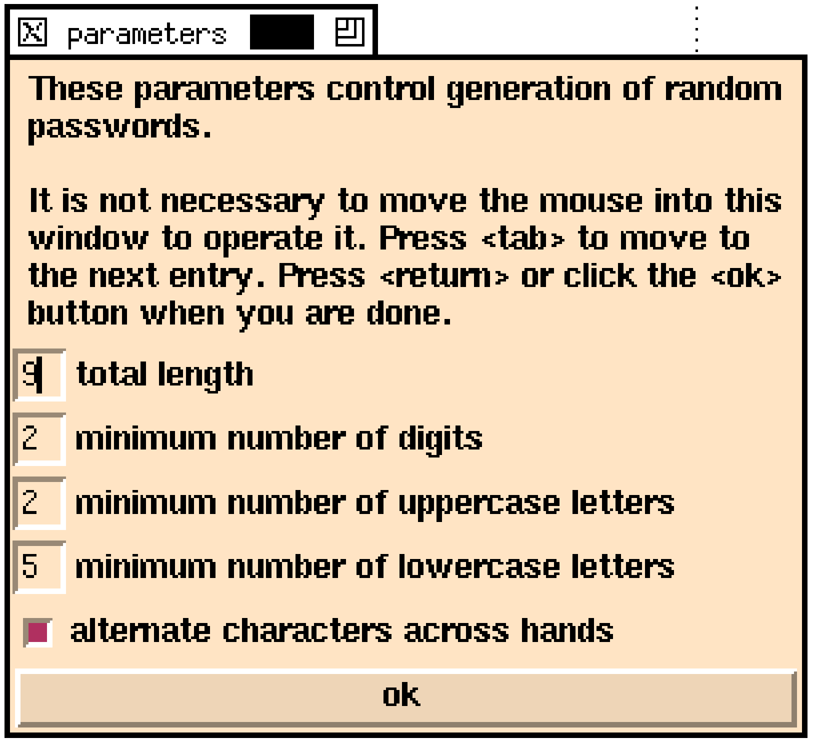
[✓ unsorted] name uid

daemon	1
sys	2
bin	3
uucp	4
news	6
ingres	7
audit	9
sync	1
sysdiag	0
sundiag	0
libes	100

Password:

9ToeM4mby

[✓] test dictionary



These parameters control generation of random passwords.

It is not necessary to move the mouse into this window to operate it. Press <tab> to move to the next entry. Press <return> or click the <ok> button when you are done.

9 total length

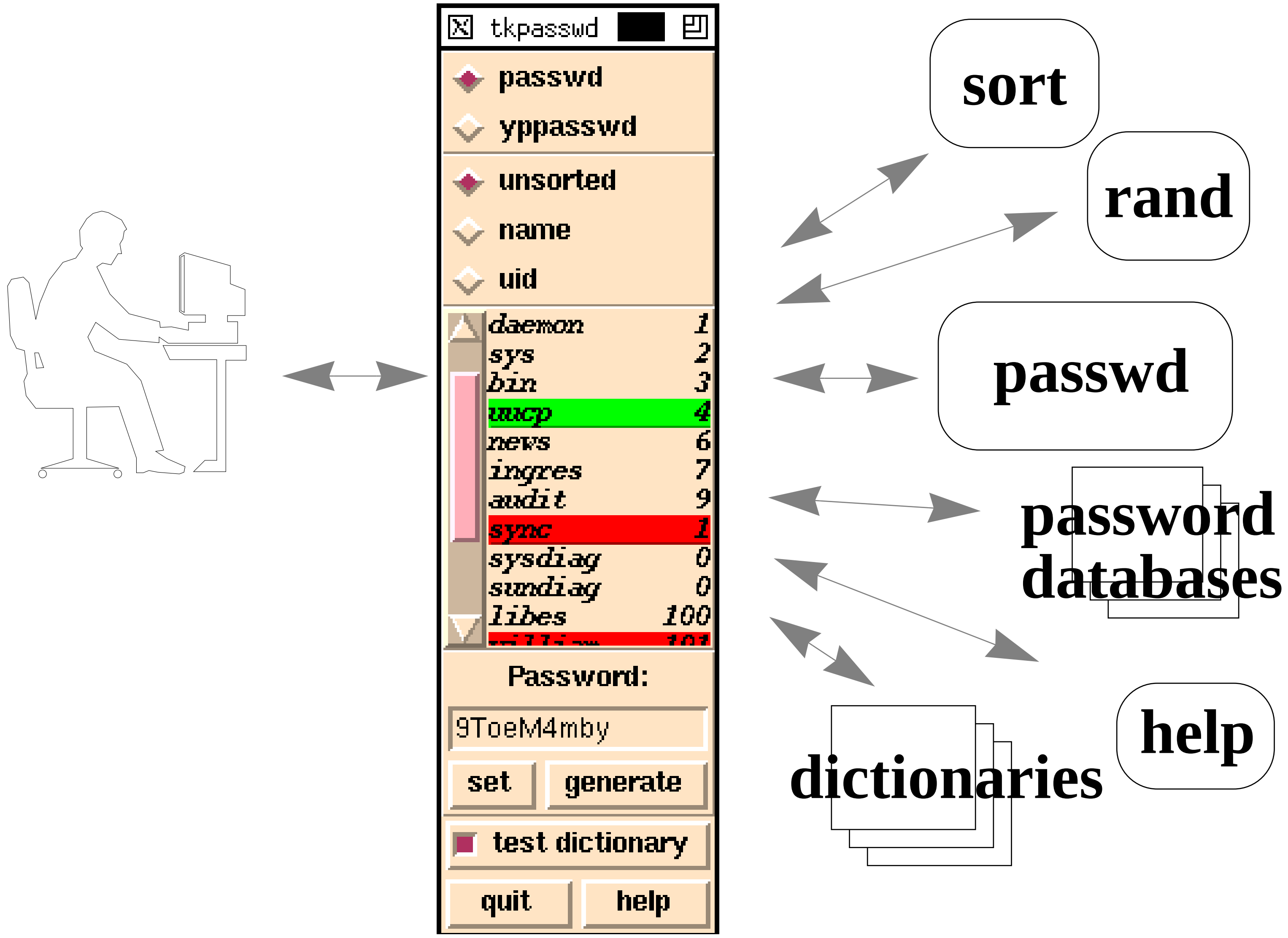
2 minimum number of digits

2 minimum number of uppercase letters

5 minimum number of lowercase letters

[✓] alternate characters across hands

tkpasswd — Performance Metrics

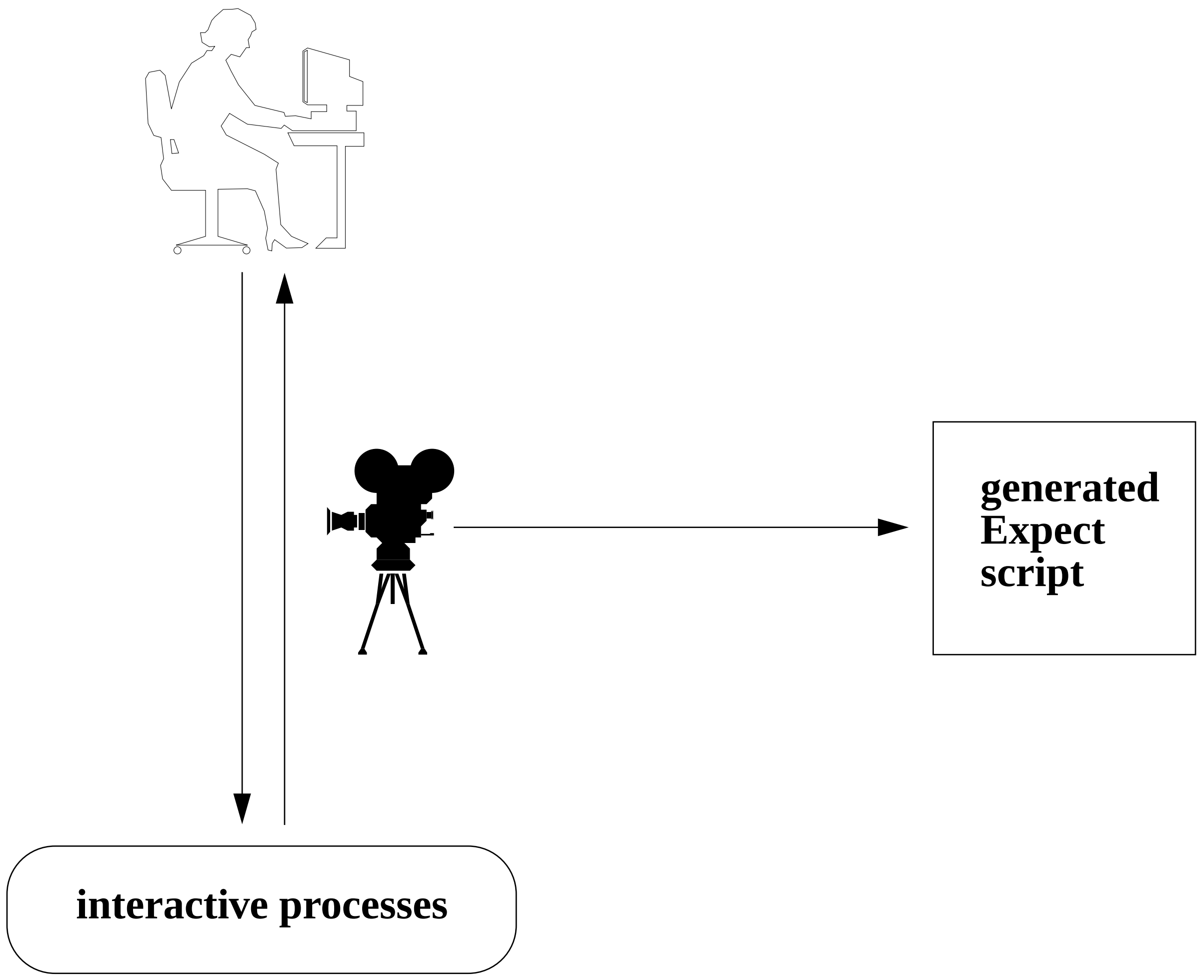


```
daemon      1
sys         2
bin         3
uucp        4
news        6
ingres      7
audit       9
sync        1
sysdiag     0
sundiag     0
libes     100
```

Password: 9ToeM4mby

How To Avoid Learning Expect

- Autoexpect – Watches you interact and generates an Expect script



Usage

- Similar in feel to the “script” command:

```
% script
script started, file is typescript
    ... interact ...
script done, file is typescript
%

% autoexpect
autoexpect started, file is script.exp
    ... interact ...
autoexpect done, file is script.exp
%
```

No guarantees

- Autoexpect has to guess about certain things
    - Timing
    - Changing Behavior
- Occasionally it guesses wrong
    - But these spots are easy to fix
- It actually does a very good job
    - has some neat heuristics
    - even experts use it
- Good news
    - very easy to use
    - free & well documented
    - nothing easier
- Bad news
    - You have to stop what you’re doing

Character graphic automation

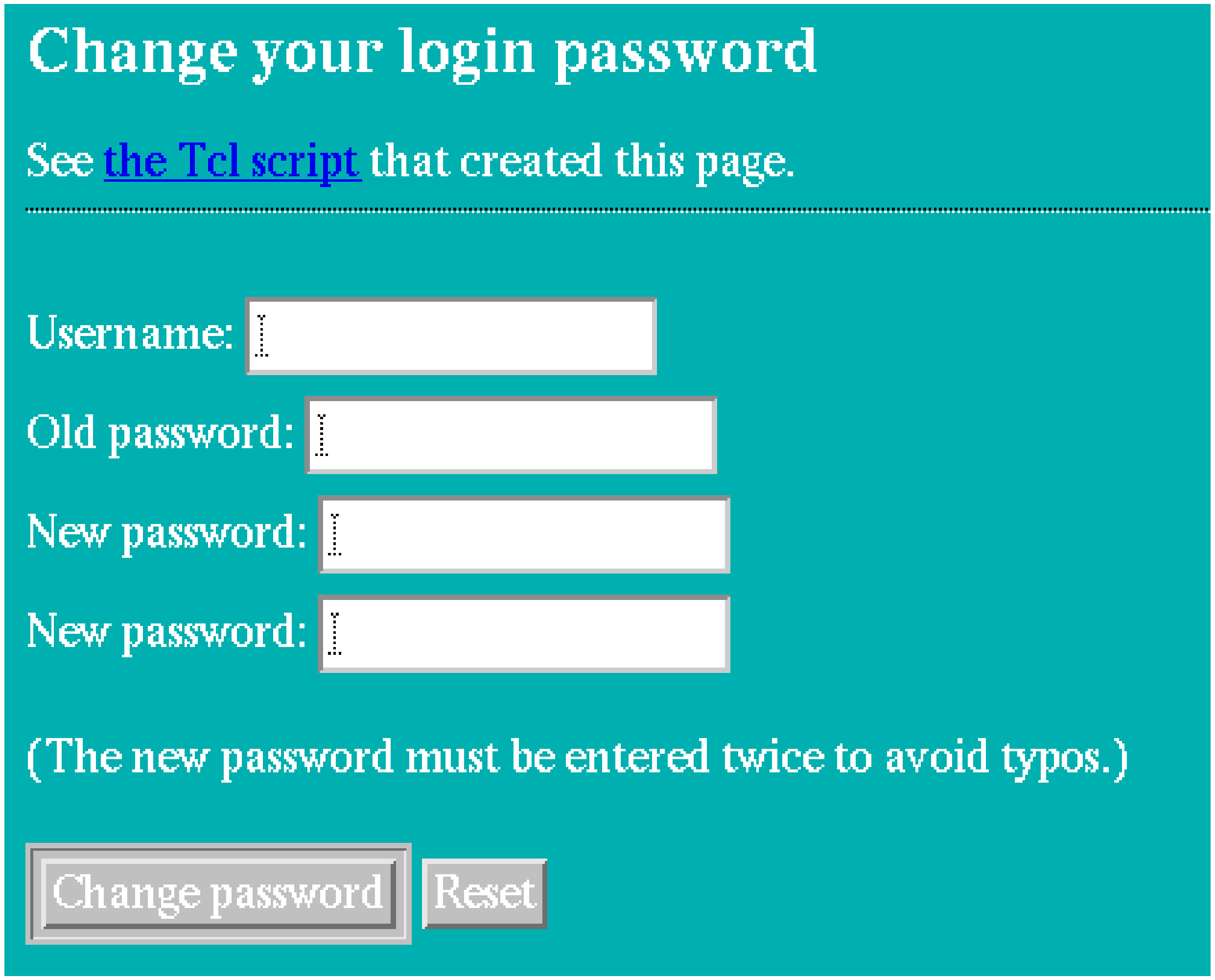
- xterm
  - send: output appears in xterm for user to see
  - expect: reads user keystrokes
  - xterm itself takes care of everything else: character graphics, select, mouse, scroll
  - good for applications that demand an xterm
- tkterm
  - Expect and Tk take care of everything
  - essentially a terminal widget
  - with enhanced expect command
  - good for Curses-based applications
- For most applications, tkterm is the way to go
  - much more flexible

Background

- Expect works fine in the background
- cron, at, batch
- CGI
- Telnet daemon
- Storing passwords in scripts
- Not storing passwords in scripts
- Storing passwords in scripts anyway

CGI

- http://expect.nist.gov/cgi.tcl



Change your login password

See the Tcl script that created this page.

Username:	
Old password:	
New password:	
New password:	

(The new password must be entered twice to avoid typos.)

Backend CGI Script

```
cgi_title "Password Change Acknowledgment"
cgi_import name
cgi_import old
cgi_import new1
cgi_import new2
spawn /bin/passwd $name
expect "Old password:"
send $old
expect "New password:"
send $new1
expect "New password:"
send $new2
puts "Password Changed!"
```

Passwords in Scripts

- Generally bad
  - but so useful
  - easy to avoid in Expect

Prompt at start-up

```
# prompt user for password
set password [getpass]

# go into background
if {[fork]} exit
disconnect

# everything hereafter is in background

sleep ...
spawn telnet ...
expect ...
send ...
```

Variations

- Example: password unknown to script or changed

```
while {1} {
    send "$password\r"
    expect {
        "sorry" {
            find-human
            set passwd [getpass-from-human]
        }
        "$prompt" break
    }
}
```

Telnet Daemon

- Secure scripts can be done with file permissions
- More secure with physically secure machines
  - put in locked room
  - turn off all daemons
- Expect script as telnet daemon
  - Users telnet to secure machine which then supplies password for them

Widely Used

“Expect has become a necessary tool for system administration. In a short time, we have used Expect in six areas and have cut out seven hours a week in tedious and repetitive tasks.”
—Thomas Naughton, Hull Trading Company

“Expect is a lifesaver for a project that I am currently involved with. I have only been working with Expect for the last couple of days, but it has already shaved about 6 months off of the completion time of the project.”
—Ron Young, System Computing Services, University of Nevada

“Thanks for making my life easier. This program has really helped me shorten the cycle time for software Q.A. Expect is like a dream come true for automation. My productivity has really increased.”
—Brian F. Woodson, 3Com NSD Software Q.A.

“Thanks for Expect. It just made an impossible project possible.”
—Bruce Barnett, GE Corporate Research and Development Center

“I figure we saved about $35K last year (Jan-Dec94) that was directly attributable to Expect. The indirect benefits drive that figure to more like $75K.”
—John Pierce, Chem Dept, UC San Diego

Expect Is Freely Available

- Easy to get
  - Cost: Free
  - URL: http://expect.nist.gov
- Easy to install
  - Portable
  - UNIX: GNU-style configure
  - Windows: ports from Cygnus and Berkeley
  - Mac: sorry
- Well documented
  - Numerous published papers
  - Comprehensive man pages
  - *Exploring Expect* (O’Reilly), ISBN: 1-56592-090-2
- Commercial Support Available
  - Scriptics
  - Cygnus Software
  - Computerized Processes Unlimited