

Expect

- a program to control interactive applications
 - it maybe possible to control interactive applications from other scripting languages as well
 - python, perl
- expect programming is *very* easy compared to perl!
 - *crackers* use expect to try and gain access to machines via ssh, ftp, telnet
 - ask AJC Blyth

Expect example uses

- as system administrators we may have need for automating interactive activities
 - testing remote access works (telnet, ftp)
 - down loading ftp files
 - changing password
 - running security checks
- we could build a script which test that our server is safe
 - from guest accounts
 - no telnet access is allowed
 - no ssh from lab machines is allowed
 - but ssh from privileged machines is working

Expect and examples

- automatically creating passwords
 - can be done via C
 - takes a lot of time to write such programs
 - will it work with NIS, shadow passwords or Kerberos?
- with expect we run the user program `passwd` and send keyboard input

Password Trial

- *advice* first run the program that you want to connect to expect by hand
 - note the output and build expect around it
 - so let us change a password for *bob*
- ```
passwd bob
Enter new UNIX password:
Retype new UNIX password:
passwd: password updated successfully
```

  - passwords were entered, but obviously the `passwd` program did not echo back our input!

## Trial results

- we note that the program prompted us for a password twice
  - both times it ended its sentence with password:
  - as we are lazy we can wait until we see password: and ignore anything that was before
    - why is this good practice?
    - what must we watch out for?

## Trial results

```
#!/usr/bin/expect
spawn passwd [lindex $argv 0]
set password [lindex $argv 1]
expect "password:"
send "$password\r"
expect "password:"
send "$password\r"
expect eof
```

## Testing the example

```
./expl.exp bob 123
spawn passwd bob
Enter new UNIX password:
Retype new UNIX password:
passwd: password updated successfully
```

- here we change user *bob* password to *123*

## Script explanation

```
#!/usr/bin/expect
spawn passwd [lindex $argv 0]
set password [lindex $argv 1]
expect "password:"
send "$password\r"
expect "password:"
send "$password\r"
expect eof
```

## Script explanation

- `#!/usr/bin/expect` script is interpreted via `expect` located in directory `/usr/bin`
- `spawn passwd [lindex $argv 0]`
  - run the program `passwd bob` and connect `expect` to this program
  - note that `[lindex $argv 0]` resolves to `bob`
    - actually argument one (hmm..)

## Script explanation

- `set password [lindex $argv 1]`
  - defines a variable `password` and sets it to `123`
- `expect "password:"`
  - waits for the program `passwd` to issue `password:` before continuing
- `send "$password\r"`
  - sends the users password to `passwd` followed by a carriage return
- note the script repeats the last two commands, why?
- finally `expect eof` wait for `passwd` to finish

## Anchoring

- you might want to match text at the beginning or end of a line, this is via
  - `^` for the beginning of a line
  - `$` for the end of a line
- also note that `*` means any number of characters

## Pattern action pairs

```
#!/usr/bin/expect
set timeout 15
expect "hi" { send "You said hi\n" } \
 "hello" { send "Hello to you\n" } \
 "bye" { send "Goodbye\n" } \
 timeout { send "I'm fed up\nbye\n" }
```

- if we run the script as shown below and type nothing we get:

```
I'm fed up
bye
```

## Pattern action pairs

- note that different actions can be associated with different input
  - note also that the default timeout time is set at 10 seconds to disable the timeout facility
  - `set timeout -1`

## Autoftp and expect

- so far we have built the front end to autoftp
  - scans the input file for URLs
  - handles arguments
- we will use expect to control ftp, we will build this up this utility
- firstly we will ftp manually

## Ftp session

```
fred@merlin:$ ftp guenevere
Connected to guenevere.
220 guenevere FTP server (Version wu-2.6.0(1))
Name (guenevere:fred): anonymous
331 Guest login ok, send your complete e-mail
331 address as password.
Password:nobody@nowhere.com

230>Welcome, archive user anonymous@merlin !
230-
230-The local time is: Mon Feb 12 15:18:14 2001
230-
230 Guest login ok, access restrictions apply.
Remote system type is UNIX.
Using binary mode to transfer files.
```

## Ftp session

```
ftp> dir
200 PORT command successful.
150 Opening ASCII mode data connection for /bin/ls.
total 20
d--x--x--x 2 0 0 4096 Aug 10 2000 bin
d--x--x--x 2 0 0 4096 Aug 10 2000 etc
d--x--x--x 2 0 0 4096 Aug 10 2000 lib
dr-xr-xr-x 4 0 0 4096 Sep 28 22:17 pub
-rw-r--r-- 1 0 0 346 Aug 10 2000 welcome.msg
226 Transfer complete.
ftp> quit
221-You have transferred 0 bytes in 0 files.
221-Total traffic for this session was 1182
221-bytes in 1 transfers.
221-Thank you for using the FTP service on
221-guenevere. Goodbye.
```

## Simple semi automated ftp script

- this script will log us into a site and then interact with the user

```
#!/usr/bin/expect

set site [lindex $argv 0]
spawn ftp $site
expect "Name"
send "anonymous\r"
expect "Password:"
send "nobody@nowhere.com\r"
interact
```

- note the `interact` statement
  - connects the keyboard to the ftp program
- most ftp servers do not check for a valid email address!

## Expect control constructs

- expect uses extends the Tcl language
- expect provides: `if`, `then`, `else`, `while`, `set`, `interact`, `expect`, `for`, `switch`, `incr`, `return`, `proc`

```
if {$count < 0} {
 set total 1
}
```

## Expect control constructs

```
if {$count < 5} {
 puts "count is less than five"
} else {
 puts "count is not less than five"
}
```

- you *must* place the statement begin brace on the same line as the `if` or `else`
  - if in doubt follow the templates in these notes

## More complex if

```
if {$count < 0} {
 puts "count is less than zero"
} elseif {$count > 0} {
 puts "count is greater than zero"
} else {
 puts "count is equal to zero"
}
```

## While statement

```
#!/usr/bin/expect
set count 10
while {$count > 0} {
 puts "the value of count is $count"
 set count [expr $count-1]
}
```

- note the effect of the braces in `{ $count > 0 }`
  - they defer evaluation of `$count`
  - if you remove the braces then `$count > 0` is internally replaced via: `10 > 0`

## for command

- has the syntax  
`for start expression next {`  
  
`}`

```
#!/usr/bin/expect
for {set count 10} {$count > 0} {incr count -1} {
 puts "the value of count is $count"
}
```

## Expressions

- 0 is false, 1 is true
- boolean operators `||` (or), `&&` (and), `!` (not)
- comparison operators `<=`, `=`, `!=` etc
- the brackets `[ ]` give this expression a higher precedence

## Expressions

- so in the example

```
set count [expr $count-1]
```

- the `[ expr $count-1 ]` is evaluated before the `set!`

## proc and return

```
#!/usr/bin/expect

proc mycompare {a b} {
 if {$a < $b} {
 puts "$a is less than $b"
 return -1
 } elseif {$a > $b} {
 puts "$a is greater than $b"
 return 1
 } else {
 puts "$a is equal to $b"
 return 0
 }
}

set value [mycompare 1 4]
puts "comparison returned $value"
```

```
./exp6.exp
1 is less than 4
comparison returned -1
```

## Autoftp Tutorial

```
#!/usr/bin/expect
this program is called exp7.exp
proc connect {} {
 expect {
 "Name:" {
 send "anonymous\r"
 expect {
 "Password:" {
 send "nobody@nowhere.com\r"
 expect "login ok*ftp>"
 return 0
 }
 }
 }
 }
}
timed out
return 1
}
```

## Autoftp Tutorial

```
set site [lindex $argv 0]
spawn ftp $site
while {[connect]} {
 send "quit\r"
 expect eof
 spawn ftp $site
}
send "binary\r"
send "cd [lindex $argv 1]\r"
send "get [lindex $argv 2]\r"
send "quit\r"
expect eof
```

## Laboratory session

- work on your assignment
- try out the following expect example
  - you will need to add your password at the position **yourpassword**

## Laboratory session

```
#!/usr/bin/expect

log_user 0
spawn pwd
expect -re "(^.*/*.*)$"
set localdir $expect_out(1,string)
expect eof

log_user 0
spawn ssh moppsy.comp.glam.ac.uk
expect "assword: "
send "yourpassword\r"
expect "\\$"
send "cd $localdir"
interact
```