Great Expectations

No need to step through multiple menus — Chris Bidmead uses Expect scripting.

Last month, we took a preliminary look at the scripting language Expect, designed by its inventor, Don Libes, to 'cure those uncontrollable fits of interaction'. The interaction I am currently trying to cure is the need to step through multiple menus when I telnet to my router in order to switch from one ISP to another.

The Expect script we have come up with so far;

```
!/usr/bin/expect
spawn telnet router
expect "Password: "
send "6666\r"
expect "Menu"
send -- "11\r"
expect "Edit:"
interact "99" { send_tty↙
 "\n" ; exit }
```

(↙ *code string continues*)

gets me some of the way by setting up the telnet session and putting me into the relevant menu (menu 11) presented by the router's software.

As I pointed out last month, the final line offers a quick exit instead of having to back out through the tiers of menus.

Yes, I know this is specific to the ZyXel Prestige 28614 ISDN router but it's a way of airing some Expect principles. Once you've grasped them you'll be able to knock up useful, short Expect scripts of your own. And given the nature of interactivity, these too will probably be highly application specific. This is quick and dirty scripting, not bombproof elegance.

You'll be able to knock up useful, short Expect scripts of your own

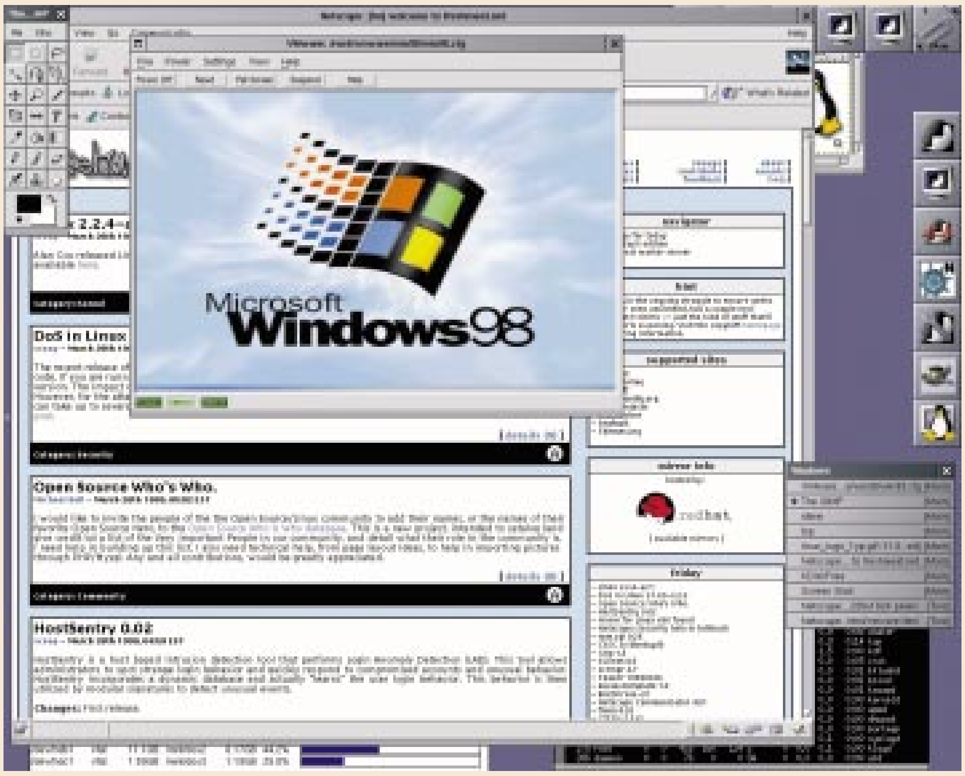
The next step is to examine the remaining interactivity and see how much of this can be scripted. Menu 11 presents me with a choice of four ISPs, one of which will be currently active. To switch off that ISP, I enter its menu number to arrive at Menu 11.1 — the Remote Node Profile for that ISP [Fig 1]. As you can see, the second item in this new menu toggles the Active flag on and off for this particular remote node. I reach this with a single down-cursor keystroke and make the toggle with the space bar. Now I need two up-cursor keystrokes and a carriage return to exit from this menu and return to Menu 11.

From there, I need to select the new ISP and go through this same routine again in order to toggle it on. In Expect, the keystroke sequence which follows my selection of an ISP by entering a number between 1 and 4, would look like this:

```
send -- "\r\033\[B \033\[A\↙
033\[A\r"
```

(↙ *code string continues*)

The first \r is the general UNIX convention for representing a carriage return. Similarly, \033 is Escape and this and the following two characters \[B comprises the standard VT100 terminal escape sequence for Cursor Down.



BOOT WINDOWS 98 UNDER LINUX

Here is a terrific screenshot which Linux gamesplayer Bradley Yen <bgyen@geocities.com> is showing on his web page at www.geocities.com/TimesSquare/Corner/9375/vmware.html.

The prospect of booting Windows 98 under Linux leaves me cold I am afraid, but this is your column every bit as much as it is mine, so please do write and tell me what you think.

The space which follows this activates the toggle, followed by a pair of Esc\[A sequences, which represent Cursor Up.

How did I know that these are the VT100 cursor control sequences? Well, I didn't — I cheated. Expect comes with a utility called autoexpect, written by Don Libes himself. Running autoexpect puts you in a debugging environment which logs your keystrokes and screen output to an executable file called, by default, script.exp. So, if instead of 'telnet router' I use 'autoexpect telnet router', I can carry out all the manual menu switching in the normal way but everything then gets logged to script.exp.

Once I've exited, I can run script.exp for an automated replay of what I've just done. But a more elegant use of this is to extract the keystroke sequences and prompts returned from the router and use these as the basis of a handwritten script of my own. One reason for this is that autoexpect captures everything, including a whole mess of VT100 cursor positioning escape sequences, whereas all I need are a few prompts.

My first pass at extending the original script to perform automatic toggling is shown in Fig 2. This puts together everything we have covered so far and adds a simple control structure, 'while 1', which translates as 'do while true' — that is, loop forever within the curly brackets. Inside the loop I have stuck an option for the user to exit by hitting '0'. The '\b's in the first line of the loop are backspaces, cunningly erasing the router's own prompt and substituting my own.

OK, this is still not total automation but from the 20-odd keystrokes in last month's column, I am now down to just two: one to switch off the current ISP and another to turn on the new one.

[FIG 1]

Remote node profile

```
Menu 11.1 - Remote Node Profile

Rem Node Name= DemonLTD      Route= IP
Active= Yes                 Bridge= No
Call Direction= Outgoing
                      Edit PPP Options= No
Incoming:             Rem IP Addr= 158.152.1.65
Rem Login=               Edit IP/IPX/Bridge= No
Rem Password= ********   Telco Option:
Rem CLID= N/A              Transfer Type= 64K
Call Back= N/A       Allocated Budget(min)= 0
Outgoing:                   Period(hr)= 0
My Login= elbid             Session Options:
My Password= ********         Input Filter Sets=
Authen= CHAP/PAP            Output Filter Sets=
Pri Phone #= 08452120666    Call Filter Sets=
Sec Phone #= 08453535667    Idle Timeout(sec)= 60

Press ENTER to Confirm or ESC to Cancel:
```

■ Multiple virtual machines...

Reader Richard Varney <rwv97c@Cs.Nott.AC.UK> wrote to me about what he calls a great PC emulator: 'Over the months I've noticed the occasional reference in your column to systems which will run Windows 9X in Linux. I have found a very competent one in the VMWare emulator. This system not only lets you run Windows but almost any OS in a virtual machine, and it is very fast. In theory, you could run Windows, Linux and NT in separate virtual machines under one base operating system (Linux or NT). It is a very exciting piece of software...'

I've had good reports about VMWare and if I had more space I'd probably find a way to sneak it in here. But I have to remind myself that this is the *UNIX* column, and although there is a version of VMWare that runs under Linux, it's not a particularly UNIX-like application. However, another Richard, KDE evangelist Richard Moore <rich@ipso-facto.freeserve.co.uk>, reminds me that an ingenious use for VMWare might be to run one Linux system under another Linux system as a crash-proof way of debugging low-level parts — I know many of you would be interested in using VMWare as a solution to the question 'How can I run my Windows applications under Linux?'

Richard Varney continues: 'Please reply when you have downloaded a copy.' Well, Richard, that might be some time yet. Meanwhile, if any of you do get VMWare up and running under Linux in a useful way, do drop me a line.

■ ...or just multiple machines

The solution I use for running multiple operating systems is to run them on multiple different machines, networked together. I've already talked about how I'm using Windows Terminal Server and Citrix's Metaframe software to pop-up Windows applications on any of my workstations that run X.

Windows Terminal Server, plus Metaframe, is a multi-thousand dollar solution but we've also talked about the GNU software, VNC, from the Olivetti & Oracle Research Laboratory, which you can download free with source code from the web site at www.orl.co.uk. The plummeting cost of hardware might even make VNC and a separate machine a cheaper solution than the $300 that VMWare will cost in its production edition. At the time of writing this, VMWare is in beta, and can be downloaded at no cost.

[FIG 2]

Automatic toggling

```
#!/usr/bin/expect -f
set timeout -1
spawn telnet router
match_max 100000
expect -exact "Password: "
send -- "6666\r"
expect  "Menu"
send -- "11\r"
expect "Edit:"
#send_tty "\rEnter Node to Switch: "
while 1 {
send_tty "\b\b\b\b\b\bToggle or 0 to exit: "
interact {
 1  { send -- "1\r/033\[B /033\[A/033\[A\r" }
 2  { send -- "2\r/033\[B /033\[A/033\[A\r" }
 3  { send -- "3\r/033\[B /033\[A/033\[A\r" }
 4  { send -- "4\r/033\[B /033\[A/033\[A\r" }
 0  { send_tty "\n" ; exit }
}
}
```

One of the main ways I use multiple (non-virtual) machines is to distribute the computing load across the network by running different X applications on different computers but popping-up their displays on a single workstation. With the exception of the Dell Power-Edge my hardware tends to lag a generation or two behind what the rest of you seem to be using. But I'm not jealous because UNIX allows me to operate much of the time as if the whole network were a single machine.

To try this yourself, you need only a couple of machines running any UNIX.
- Network them together and then set yourself up as a user of the same name on each.
- Now you need to tell each machine that the other is trustworthy to the extent that a user on one is equivalent to the same user on the other. This cuts through a lot of the security red tape — so tread carefully.
- You do it by creating a config file called /etc/hosts.equiv. It might look something like the following.

```
# machine names have to appear
 in here EXACTLY in the form
# they appear in the second
field of /etc/hosts, otherwise
# rlogin and rsh will demand
passwords.

nextmachine.cbidmead.home.edu
dellpe.cbidmead.home.edu
pc315.cbidmead.home.edu
ls550.cbidmead.home.edu
```

The first field of /etc/hosts contains the IP address and the second field the canonical name of the relevant host. Subsequent fields establish nicknames (my NeXT machine is also known as 'next'). The hosts.equiv file says that user bidmead (say) on my NeXT machine is equivalent to the same user on the Dell PowerEdge, the IBM PC315 and the Apricot LS550. Now if I want to send a file from one machine to another I don't need to go into ftp. If I'm working on the Dell, I can just say, for instance:

```
$ rcp myfile next:
```

And myfile will be remotely copied across the network to the bidmead home directory on the NeXT machine.

This will work for every user with the same name on each machine, except root; for obvious security reasons. The mechanism behind this allows me to login directly to another machine.

There is another way of doing this; by creating user-level rather than host equivalences with a file called .rhosts in the users home directory. This file lists not only trusted hosts but also, optionally, trusted users within those hosts, and /etc/hosts.equiv and ~/.rhosts can be used together. But I should warn you that security-conscious people do not like any of this stuff because a network of mutually trusting systems is only as strong as its weakest link.

This cuts through a lot of the security red tape

So far, we have only worked at the command line with RPC. We can extend this into X if we remember that X has its own defence mechanism — an 'access control list' that defines which other systems can use a particular X server.

Let's say I want to run Netscape on the screen in front of me (the IBM PC 315) but I want to use the copy that is on the Dell machine because the software version is newer and because the Dell hardware has a bit more *oomph*.

First, I need to tell my local X server that it is OK for the Dell to grab part of my screen. I can do this by setting up a config file but the simplest way is to do it on-the-fly with the command:

```
$ xhost +dellpe
```

This returns the message 'dellpe being added to access control list', and now we are ready to roll. Until recently, the way I always used to do this next bit was to drop into an xterm session and issue the command:

```
$ rsh dellpe /opt/netscape/✓
netscape -display pc315:0
```

(✓ *code string continues*)

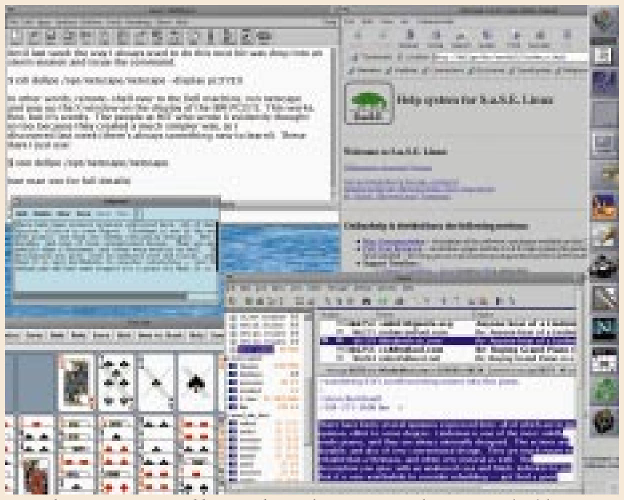
In other words, remote-shell over to the Dell machine, run netscape and pop up the X window on the display of the IBM PC315. This works, fine, but it is wordy. The people at MIT who wrote X evidently thought so, too, because they created a much simpler way as I discovered last week — there is always something new to learn!

These days I just use

```
$ xon dellpe /opt/netscape✓
/netscape.
```

(✓ *code string continues*)
— see man xon for the full details.

CROWDED HOUSE



I don't normally work with a screen this crowded but I wanted to show you the possibilities of RCP and X. Emacs, the editor I use to write this column, is running locally under Linux on the IBM PC315. In the right-hand corner, Netscape is running on the Dell PowerEdge. The FreeCell window *(bottom left)* belongs to the Apricot PC315. And Ameol, the off-line reader for Cix, is a Windows app running on the Siemens Primergy server, piped in via Citrix MetaFrame. The light blue Xclipboard window is running on the NeXT machine and this is such a handy way of copying text across machines that I've added it as an icon at the bottom of my AfterStep desktop Wharf widget; the column of icons running down the right-hand side of the screen.

PCW CONTACTS

Chris Bidmead welcomes your comments on the Unix column. He can be contacted via the PCW editorial office (address, p10) or by email at unix@pcw.co.uk