Slap, bash and tickle

Expect a bit of a Tcl from Chris Bidmead on scripting languages.

Unix is awash with computer languages, each approaching the business of designing programs or automating tasks from different philosophical positions. Today, I'd say if you were only going to learn one, learn Perl, but of course you're not only going to learn one. If you have embraced the spirit of UNIX to any extent at all you are also going to want to dabble at least in bash, procmail, Tcl, awk, sed and... well, it's hard to know where to stop. And those are just the scripting languages.

The inventor of Tcl, John K. Ousterhout, maintains that over the past 15 years there has been a fundamental shift away from system programming languages such as C toward scripting languages like Perl or Tcl. He believes that in the years to come, the latter type will handle many of the programming tasks of the next century better than the former. To view his full half-hour argument, see his paper *Scripting: Higher Level Programming for the 21st Century* at www.scriptics.com/ people/john .ousterhout/scripting.html.

Although this column doesn't set out to be a programmer's workshop, in the past we've made the odd recce into awk, sed, bash and procmail. The idea has been to whet your appetite and give you a flavour of how these languages help you to take control of your computing environment. In much the same spirit I want to venture this month into the scripting language called Expect.

Expect is an extension of Tcl, so I suppose the logical thing would be to start with that. Tcl is a 'task control language' and you get extra brownie points for calling it 'tickle', which is the way Ousterhout pronounces it (although T-C-L is fine, he says).

Tcl is used with its companion graphical toolkit Tk as a fast way to build GUI-based applications and utilities but Tcl's library of functions is also used by Expect for a rather different purpose best summed up by Expect's author, Don Libes, as 'Curing those uncontrollable fits of interaction'. This was the title of the paper with which Libes introduced Expect to the 1990 USENIX Conference in Anaheim, California and may require some explanation for those of you approaching all this from the direction of Redmond or Cupertino.

Microsoft software, borrowing heavily from the Macintosh, likes to sit around waiting for user input. Unix software traditionally gets on with things behind the scenes once the user has set it going and doesn't come back until the job's done. The job may involve running multiple, separate programs and Unix passes data between these without user interaction via pipes, or by using command substitution.

By way of illustration, Libes cites a Unix utility which doesn't work that way. Passwd is the utility for setting up and changing passwords and if you're used to the Unix way of doing things you might guess it would operate something like this:

```
$ passwd -o onion -n parsnip
```

where the -o flag introduces the old password and -n indicates the replacement.

This would lend itself to automating the process of password changes, for example with a cron script that ran once a month and updated the password of each user with a new one made up of a modified word randomised from a dictionary. The cron script would then email the user with the new password, so that next time he or she logged on... but I suspect you're already starting to spot the recursive flaw in this approach, to say

WHAT'S UNDER THE BONNET?

'Open source' is the buzzword, but many non-programmers have never even seen source code. Well, even if you're not a car mechanic it's still worth lifting the bonnet once in a while. If you want to do the equivalent with the code that runs your Linux, FreeBSD or other open source system, check out this site at www.openresources. com

▸ **Fig 2 Delve deeper at expect.nist.gov to dig up a little unexpected humour**

> Expect is totally totally a tool for automating interactive applications such as telnet, ftp, passwd, fsck, rlogin, tip, etc. Expect totally makes this stuff trivial. Expect is totally totally also useful for testing these same applications. Like, gag me with a spoon! Like, oh my gawd! Like, gag me with a spoon! And by adding Tk, you can also wrap interactive applications in X11 GUIs.
>
> OH! Like, oh my gawd! I almost forgot to tell you! Like, oh my gawd! Like, there is totally like this bitchin' sale goin' on at like The Merry Go Round this week and like I went there and picked up a few minis and this totally bitchin' hat and charged it on Daddy's card. Everything was like, you know, going fine until this totally skanked-out saleslady tried to get me to buy this mondo grody belt. I was like, you know, like "gag me, lady, totally don't make me barf!" Anyway... So she like rung me up and... oh nevermind, I'll tell you later.

nothing of the security problems. Which is why the passwd utility doesn't work as above but instead insists on eating the new data directly from the user's hand via an interactive dialogue.

Libes points out that once a utility is set up to work like this, traditional scripting as a method of extending that utility fails. For example, you wouldn't be able to write a script that vets the user's input into the passwd program and which warns against passwords that are too easy to crack. The plus side is that traditional scripting cannot create a passwd wrapper that watches what the user enters and squirrels that data away for future use in cracking the system.

Libes' Tcl extension, Expect, solves the problem (and opens that can of worms). Expect is so-called because its key feature is to wait and watch for the occurrence of particular strings and then carry out actions accordingly. This makes it possible to swallow interactive programs and treat them like more traditional UNIX-style utilities.

These languages help you take control of your computing environment

Let's examine how this works. My ZyXel ISDN router is effectively a small UNIX machine on the network dedicated to routing TCP/IP packets in and out of my local area network. It allows me to switch between up to four different Internet Service Providers, and the interface that controls this is a Telnet connection.

I Telnet to the router from whichever workstation I happen to be at, enter the password that guards the router from intruders and up pops a menu in the Telnet window. From the Zyxel's main menu I can do a number of things like put the router into diagnostic mode, read its error log, define which machine on my network SMTP mail should go to and so forth. But mostly I use the Telnet session for switching around between different ISPs.

The Unix way of doing this would be with a small utility called, say, isp which took a command line parameter. So,

```
$ isp -i demon
```

would switch me from whichever ISP I happen to be currently using, to Demon Internet.

By the way, I know from your emails that many of you are bemused by this kind of approach and see operations at the command line as a return to the Dark Days of DOS. I'd better assure you that once you have a utility like isp in place it's a trivial matter to set it up under, say, the AfterStep GUI to launch elegantly from an icon if that's what turns you on. As an added refinement you could even set the icon to reflect the current ISP connection, although that's a little more complicated.

Unfortunately, isp doesn't exist and the Telnet session I have to go through to switch ISPs involves several series of interactive keystrokes. Yup, it's one of those 'uncontrollable fits of interaction.'

Here's the sequence:

1. Telnet the router
2. Respond to the Password: prompt
3. Go to Menu 11
4. Select the ISP setup that's currently switched on and switch it off
5. Select a different ISP setup and switch it on.

As this was my first encounter with Expect I decided to approach the problem in stages. To start with, it would be useful just to automate the first three steps, to pop up Menu 11, change ISPs manually and then exit.

Here's the Expect program which does this:

```
#!/usr/bin/expect
spawn telnet router
expect  "Password: "
send  "6666\r"
expect  "Menu"
send ñ "11\r"
```

Self-explanatory, really. Spawn a process that Telnets the router, wait until the string "Password: " turns up, then send the password (which happens to be "6666) followed by a carriage return (\r"). Then wait for the prompt asking for a "Menu" and send the string to get to Menu 11.

And that's exactly what this script does. Unfortunately, having carried out these instructions to the letter, the program then ends, killing the spawned Telnet session and dropping me back at the command line. This is not very useful, so I need to add another couple of lines:

```
expect "Edit:"
interact
```

Virtually plain English again. Wait until the router's menuing system prompts me on which ISP entry to edit and then drop me interactively into the Telnet session.

Let's add one more refinement. To exit from this once I've switched ISPs I have to wind back down through two menus because that's how the ZyXel's menuing system does it. But I can give the interact command a couple of parameters, as in Fig 1. This says: enter an interactive session with the currently spawned process but keep an eye open for the number 99; if the user enters this, exit, closing the Telnet session; at the same time, send a line-end to the window in which we've been operating, just to keep things tidy.

[FIG 1]

```
interact "99"   { send_tty "\n" ; exit }
```

This simple program saves me about 20 keystrokes every time I switch ISPs. It

still leaves some clunky interactivity once I'm in menu 11, which will give us an excuse to venture further into Expect next month.

Meanwhile, if 'man expect' doesn't give you the man pages you, er, expect, you can pop off to the Expect home page at http://expect.nist.gov for a copy of the software and documentation [Fig 2].

Junkbuster

There's a nice little poem by Ogden Nash about the proliferation of roadside advertisements in the fifties:

I think that I shall never see
A billboard lovely as a tree.
Perhaps, unless the billboards fall,
I'll never see a tree at all.

Today, on the internet, advertisement banners are what pay for much of the free webbery that is becoming part of our daily lives.

I've heard it argued that it's our duty to read these banners because opening the web pages that carry them implies our agreement to do so. In any case, the argument goes, advertising is informative and we owe it to ourselves to stay informed. Other people call this stuff junk. Most people, I suspect, don't care.

As the name implies, the Junkbuster folk at www.junkbuster.com are among those who believe that you shouldn't have to tolerate uninvited ads. The free (that's 'free' as in GPL, not free as in 'FREE!!!') software they provide is a great way of keeping your screen clean and I use it a lot of the time.

The Internet Junkbuster Proxy not only blocks unwanted banner ads but can also, claims Junkbuster, 'protect your privacy from cookies and other threats', including blocking out known rude sites you might not want your offspring to stumble across.

It runs under most Unixish operating systems — including, of course, Linux for which there are rpm distributions — and also under Windows 95/98/NT. If you're on a local area network you can put it on any of the machines and make sure all the local browsers point to it.

My Junkbuster is currently installed on the Dell PowerEdge, and my Netscape Edit/Preferences/Advanced/Proxies are all configured to point to dellpe.cbidmead.home.edu:8000. When the Junkbuster proxy server receives a request from any of the browsers it redirects it out to the internet but monitors the response against its configuration files. These config files list the sites to block, or directories within sites, or even the names of individual images, and the masks can include regular expressions.

▲ FIG 3 THIS IS THE VERY USEFUL NEWSBYTES WEB PAGE AS IT LOOKS OVER A DIRECT CONNECTION (BOTTOM LAYER), HOW NETSCAPE DISPLAYS IT ONCE JUNKBUSTER HAS STRIPPED OUT THE ADVERTISING BANNER (MIDDLE LAYER), AND HOW STEFAN WALDHERR'S MODIFICATION TO JUNKBUSTER IMPROVES THE LOOK BY SUBSTITUTING A 1 X 1 PIXEL TRANSPARENT IMAGE WHICH IS JUST VISIBLE IN THE PICTURE BECAUSE IN THIS CASE THE PAGE WRAPS A BLACK BORDER AROUND IT

By default, the blocked banners are replaced by your browser's own 'Missing Image' icon [Fig 3] which you might think looks a bit naff. The Junkbusters say they've had many requests about how to avoid this, and on the one hand their official response is: 'Apart from making it harder to catch unintended blocking, this might also displease the owners of the page, who could argue that such a change constitutes a copyright infringement.'

On the other hand, they believe that 'merely failing to allow an included graphic to be accessed would probably not be considered an infringement: after all this is what happens when a browser is configured not to load images automatically.'

If this copyright argument seems cogent to you, you'll probably want to set your proxy to block visits to Stefan Waldherr's site at www.waldherr.org/junkbuster, where he offers a slightly modified version of Junkbuster which automatically replaces images with a 1 x 1 pixel transparent GIF. This is the version I use. Tut, tut...

PCW CONTACTS

Chris Bidmead welcomes your comments on the Unix column. He can be contacted via the PCW editorial office (address, p14) or by email at unix@pcw.co.uk