by Peter Collinson, **Hillside Systems**

JANE MARINSKY

xterm

In the wake of my article on `rlogin` ("The rlogin Program," *SunExpert*, December 1998, Page 22, `http://sw.expert.com/C2/SE.C2.DEC.98.pdf`), someone sent me an email suggesting I do an article on `xterm`. "Good idea," I thought. However, the original email from my correspondent has disappeared, so I cannot credit him publicly, sorry.

I use the Common Desktop Environment (CDE) on my Solaris system, but I still prefer to type into the machine using the `xterm` program. There are a couple of reasons for this. First, you can run `xterm` using different foreground and background colors, and you can also set the color of the text cursor to complement your choices. I use colors when I log into different machines so I can easily tell which window contains a connection to which machine. Second, I find the cut-and-paste model provided by `xterm` to be the "right way" to do things; it has minimum mouse interaction for maximum effect.

If you want to use `xterm` under CDE, then you'll have to work a little to find it. It lurks, seemingly unappreciated, in `/usr/openwin/bin`. Assuming your search paths are established to include this directory, you can type `xterm` into a terminal window and get an invocation of the program running. I've used the Create Action tool to make a clickable icon on my desktop that runs `xterm` for me. Spending a little time creating an action is worth it because you can then add the icon to the control panel's pop-up menus, saving time later.

The `xterm` program is a terminal emulator supporting the Digital Equipment Corp. (DEC) standard VT100, VT102, VT200 and VT300 programming models for terminals. It also supports Tektronix 4014 terminals that use storage screens to display vector graphics. I suppose there are numerous people out there who've never come across a storage tube terminal, so a little digression into history is in order.

In the early days of graphical output from computers, several companies adapted the cathode ray tube technology that started in oscilloscopes. The oscilloscope uses X and Y inputs to position the cathode ray beam on the screen. The beam creates a small dot of light. By applying a time signal, the beam is moved over the screen, leaving a trail in the phosphor. The phosphor retains the image for a short period of time, but the brightness gradually decays. However, when the moving beam is coupled with the miracle of persistence of vision in humans, then the excited areas of the screen can be seen as a static image of equal brightness by the viewer.

A computer can use the same technology to move the beam and then generate a drawn image. We ended up with devices like the DEC GT40, which was a PDP-11-based graphics processor that endlessly executed a list of vector commands and generated an image on the screen. This device was undoubtedly used in early CAD/CAM applications, but is probably more noted for its

Moonlander program, which still persists in arcade games.

The GT40 needed a processor to continuously execute a sequence of commands because it used a short persistence phosphor for its display, so the image faded allowing animation and change of information. There are other phosphor technologies that can be used. The Storage Screen oscilloscope can capture a single event on the screen, storing the track of the beam in the phosphor for some appreciable time. The image either gradually decays or can be cleaned from the screen by flooding the screen with electrons.

Tektronix used this technology to create computer displays. The device could accept a nontime-critical sequence of vector commands from the computer and draw an image that was retained on the screen until it was cleared. The device was used in early UNIX systems and became a de facto standard output device for many early graphics programs. At the time `xterm` was created, it seemed natural to want to use this software, so the Tektronix emulator was born. Until recently, I used this emulator with the `plot` and `graph` programs to see a graphical representation of the electrical usage in my house. My scripts now generate PostScript.

Incidentally, an ex-colleague of mine, John Bovey of the University of Kent in England, wrote a lightweight X11 terminal emulator called `xvt` around 1992. It omits all the largely unused Tektronix code. This became popular on early Linux systems because it was fast and small. It has now moved into the hands of others and become `rxvt`.

Using xterm

The `xterm` program works in a similar way to `rlogin`. However, rather than interacting with a server on a remote machine, the program communicates information with the X server that controls your screen. The X server sends the characters that you type into the running `xterm` process as X events. The characters are in turn passed by `xterm` into an application. Output from the application is captured and sent back to the X server to be placed on the screen. The application is usually a shell running the commands that generate output destined for your screen. The shell and other programs need to be fooled into thinking that they are talking to a terminal. To achieve this transparency, the `xterm` program talks to the control end of a pseudo-terminal and the commands communicate with the “user” end of the pseudo-terminal, which provides the necessary terminal interface.

By default, `xterm` will start your favorite shell taken from your `SHELL` environment variable. It doesn’t have to; you can start an arbitrary command. For example,

```
$ xterm -e vi
```

executes the `vi` command in a new window. The `-e` parameter should be the last element on the command line and this restriction allows you to supply program arguments:

```
$ xterm -e vi filename
```

The window will disappear when the `vi` command terminates. So typing something like

```
$ xterm -e ls
```

will run the command in a new window, but it will stop quickly and you may not see it. Incidentally, the `-e` argument is supported by `dtterm`, the standard CDE terminal emulator.

Cut and Paste

I mentioned that one of the things I like about `xterm` is its easy-to-use cut-and-paste operations. The description that follows uses the default button settings. If you are left-handed, you may wish to change them. I generally feel that it’s a bad idea to change default settings that are fundamental to the use of a program because you’ll find it difficult to use other people’s desktops. You must “take a view on this,” as lawyers say.

For cut and paste, you start by sweeping out an area with the left mouse button. Point at the start of the area you want to pick up, hold the left button down and move the mouse. The area you are about to pick up will be highlighted. It is loaded into the cut buffer and can be deposited in another window (or part of the same window) at the cursor position by pressing the middle button. If you’ve included the end of a line in the selection, you’ll insert a new line when you press the middle button. The information is stored and can be inserted as many times as you like, until you reload the selection buffer.

If you double-click the left mouse button, you’ll find that a “word” is highlighted; triple-clicking will pick up a whole line. The right mouse button is used to extend any selection that you have made. It can be used in sweep mode, or just pointing to a spot further down the screen and clicking with the right button will extend the selection. Another way to create a selection area from scratch is to click at the start point with the left mouse button and then at the end point with the right mouse button.

Incidentally, it’s useful to know that you can perform cut and paste while holding down the Shift key. So if the application in the `xterm` window has used mouse buttons for its own editing application, and this doesn’t work with the normal selection buffer mechanism, you can use Shift with the mouse keys to obtain “global” cut and paste. When working with my editor, I actually find it useful to have both an application-specific cut buffer and a window-wide one.

And, oh yes, if you use certain other operating systems, you will find you can cut and paste text in and out of certain parts of your screen and not others. This, to me, is deeply annoying. It’s very unusual to find any application running under X that doesn’t permit the pasting of text into a suitable place.

Scrolling

There are several values in the `xterm` program that can be set either from the command line or from X resources to change the default behavior of the program. I generally call `xterm` with

```
$ xterm -sb -sl 2000
```

The `-sb` flag adds a scrollbar to the screen permitting access to the information that has scrolled off the screen. It's a history of what has happened, so you can scroll back up the screen to see what has happened before. The `-sl` option gives me a 2,000-line history that I can scroll into rather than the default 64 lines.

You can turn the scrollbar on and off via the `xterm` control menus. If you hold down the Control key and press the mouse buttons at the same time, you are presented with one of three menus, depending on which mouse button you have chosen. The middle button allows you to turn options on and off. Actually, I hate this kind of secret chorded keystroke, it's one of the "problems" with X.

When you have scrolled the screen looking back at the history, you will move the active last line out of sight at the bottom of the screen. By default, `xterm` will reposition the screen when it receives further data from the application. This can be annoying if you are using `tail -f` to watch a log file and have scrolled up to look at some previous activity. You can turn off this behavior by using the control menu, or by supplying an `-si` flag to the program on start-up. You can also make `xterm` automatically jump to the last line when you type something. This action is turned off by default. I find the default settings for scrolling are generally tolerable.

Putting Color on Your Screen

There are several parameters that are used to permit `xterm` to display in different colors. You can supply these from the command line, and I generally do this while I am testing for viable color combinations. For example, to set up an `xterm` with a dark blue background, white text and a yellow text cursor, you type

```
$ xterm -fg white -bg darkblue -cr yellow
```

You may also want to add the scrollbar options I mentioned earlier. The color names are taken from the `rgb.txt` file, which lives in `/usr/openwin/lib/X11` on my system, although there are symbolic links from elsewhere.

The colors can also be specified by X resource names that are somewhat more lengthy (I've wrapped the lines below for printing; when entering this text use one line, and remove the backslash characters):

```
$ xterm -xrm '*foreground: white' \
        -xrm '*background: darkblue' \
        -xrm '*cursorColor: yellow'
```

The `-xrm` option is standard with every X command and supplies a value to the X resource manager. Each X application consists of a hierarchy of named objects known as "widgets." Each widget has a set of resources that can be set internally in the program, but also from outside the program. The result is you can tailor your X application to your own needs once you understand the set of rules.

In the example above, I am using the star (`*`) operator in the same way I do when looking for files in the file system tree. It sets every widget with a named resource of `foreground`, `background` or `cursorColor` to the argument value. The command may be somewhat too overpowering because it will set the menus to the color set and you may not want to do that. I usually want to be more specific and only change the colors in the `xterm` window. The `xterm` window is called `vt100`, and because I am invoking the command using `xterm`, the application will be called `xterm`. So to only set the colors in `xterm`, type

```
$ xterm -xrm 'xterm.vt100.foreground: white' \
        -xrm 'xterm.vt100.background: darkblue' \
        -xrm 'xterm.vt100.cursorColor: yellow'
```

If you try this, you'll find that the menus are now the default color. Before trying to change that, let's make it easier to use this setup. Incidentally, you can replace the `xterm.` string at the start of each name with a star if you want to save typing.

Now that we are happy with our settings, we can load them into our `.Xdefaults` file so that we don't have to type all this stuff every time we want to invoke the command. I expect you are relieved about that.

One of my aims was to provide different color settings for the different machines I use, so when we edit our `.Xdefaults` file, we want to add lines that are tagged with a specific name. My main machine is called `craggy`, so I have the following lines in my `.Xdefaults` file:

```
craggy.vt100.foreground: white
craggy.vt100.background: darkblue
craggy.vt100.cursorColor: yellow
```

If you are using CDE, there's a button called Reload Resources in the Application Manager under Desktop Tools. When you edit your `.Xdefaults` file, you'll need to run this application to ensure your X server is given the new settings. If you are not using CDE, then you will find a command to load the resource in your start-up sequence somewhere, and you'll need to type

```
$ xrdb -load .Xdefaults
```

Once that is done, you can type

```
$ xterm -name craggy
```

and now whenever we name the `xterm` as `craggy`, it will use the settings we have established for that named application. Note that most programs read information from the resources database only on start-up, so changing some settings will only affect new invocations of the program. Also, make sure the settings work before logging off, otherwise you may log in again and find that the system is unusable.

Changing the menu colors is a little more complicated and uses some X magic. In `xterm`, menus are "pop-ups" and don't really have obvious names in the hierarchy of widgets in the application. We want to say, "set these colors whenever you start up a menu." In X-speak, we need to affect the *class* of menu objects. We do this by using a capital letter when we set the resource. So to change the menu colors for `craggy`, we'll add some text to our `.Xdefaults` file:

```
craggy.SimpleMenu*foreground: darkblue
craggy.SimpleMenu*background: yellow
```

I'm not exactly sure what exists underneath `SimpleMenu`. I don't care either. I can use the star operator to fill in the missing bits for me.

Right, I can devise several color schemes for different invocations of `xterm` and so I am ready to take the next step of running `xterm` applications on different systems. I'll leave the full horrors of this for another time.

Looking into my `.Xdefaults` file reminds me that I have some other default settings for `xterm` programs. They apply to every invocation of `xterm` because the entries in the file start with a capital X.

The first line changes my keyboard's default behavior and adds a new translation for my keyboard:

```
XTerm*VT100.Translations: \
    #override Ctrl<Key>space: string(" ")
```

This ensures that my applications are sent a space when I type Control-Space. I did this eons ago and seem to recall that otherwise Control-Space generates nothing. I found that when using my editor it was convenient to generate a space so that my somewhat inaccurate typing would find the correct character.

The second entry changes the way a word on the line is constructed when you perform a double-click to select it:

```
XTerm*charClass: 33:48,37:48,45-47:48,64:48,126:48
```

This is a comma-separated list of values. Each value is separated by a colon. The number (or number range) before the colon is a value in the ASCII character set; the number after the colon is 48 in this case, putting the characters into the same group as the alphanumerics used in words. Easy? Well, no. Dreadful.

When you select a word by double-clicking, `xterm` scans on either side of the cursor to select the characters that comprise the word. When creating the selection, it scans outwards from the cursor position (in both directions) until it finds a character that isn't in the character set indicated by, in this case, 48 in the lookup table.

It's often convenient for a word to contain some nonalphanumeric characters, so you can select file path names in one double-click action. The setup line above was suggested from the `xterm` manual page and adds exclamation mark, percent, dash, period, slash and ampersand to the list of characters that are deemed to be a part of a word. Recently, I've added tilde to this list (126).

Setting Banners

The `xterm` program can set the window title bar so you can place useful information into it. I put the name of the machine and the current working directory in the title bar. I've published this tip before, but it always seems to provoke interest so I don't feel too bad about printing it again. (Incidentally, this also works with `dtterm`.)

You send standard terminal setting sequences to the running `xterm` program, causing it to change the title bar. The sequences form a private extension to the ANSI standard, so they will be accepted on any ANSI terminal and should be discarded when not recognized.

The sequences involve control characters and I will use the `echo` command notation to show this. So in the sequences below, `\033` is the single character Escape and `\007` is the character Control-G. You will need to quote them to your editor to get them inserted into the file.

In order to load a string into the title bar for X, you will need to send

```
\033]2;string\007
```

That's Escape, a closing square bracket and the digit "2" at the start of the line. You can also try this out with the `echo` command:

```
$ /bin/echo -n '\033]2;Hello\007\c'
```

I'm using `/bin/echo` because it supports the use of escape sequences. The built-in `echo` command in your shell may not. You can type `\033` to get the Escape character sent by the command and `\007` to get Control-G. The `\c` at the end of the string means that `echo` will suppress the new line that it normally appends to the end of its output. We don't want that because we are sending a sequence that is interpreted by the terminal emulator. If you now type this into your shell, you should see the title bar set to "Hello."

If you replace the "2" immediately before the semi-colon in the sequence above with "0," then the string will change the title and iconic name of the window. Also, you can change just the icon name by using "1."

You can now easily make a private command that will retitle your screen. Simply create a file called `retitle` in your private `bin` directory, which contains the following:

```
#!/bin/sh
/bin/echo '\033]2;'$*'\007\c'
```

and off you go. The `$*` shown above is replaced by the arguments to the script. Putting the escape sequence into a `cd` command takes quite a bit of explanation, which I don't have space for here.

The best source for information about `xterm` is the manual page, which, to be honest, is not exactly easy going. However, nothing ventured, nothing gained.

***Peter Collinson** runs his own UNIX consultancy, dedicated to earning enough money to allow him to pursue his own interests: doing whatever, whenever, wherever... He writes, teaches, consults and programs using Solaris running on a SPARCstation 2. Email:* `pc@cpg.com`.