CHAPTER 5

The Cmdtool/Shelltool Terminal Emulator

There are many "terminal emulator" programs available for The X Window System. This chapter shows you how to use the OpenWindows standard terminal emulator, a program known as *cmdtool*. *Cmdtool* is a general-purpose terminal emulator with a variety of features such as scrolling, logging, history editing, and so on. And unlike some other programs, such as the MIT version of *xterm*, *cmdtool* is compliant with the OPEN LOOK specification. and is therefore very similar to the editing windows provided by most OPEN LOOK-conforming applications. The name *shelltool* invokes the same program in a different mode.

This chapter also discusses the OpenWindows text editor, *textedit*, since it shares many features with *cmdtool*†.

Like any other terminal emulator such as *xterm*, these tools provide a window that works much like a standard computer terminal, and that lets you do anything you can do with a standard terminal. Each such window is connected to a *shell* or command interpreter program (normally the same shell that you use for your login shell), so you can run shell commands directly. Once you have one terminal emulator window on your screen, you can use it to start up any other clients you need, including other terminal emulators. You can run as many terminal emulators as you need. For example, you might be examining or editing a program's source code in one *cmdtool* window, while compiling or debugging it in another.

The easiest way to start a new *cmdtool* is from the *Workspace->Programs* menu. As it is the default "Program," you need only click SELECT on the *Programs* menu to start an additional *cmdtool* at any time.

† If you aren't using OpenWindows and wish to get a copy of *cmdtool*, you can FTP it (along with the XView toolkit, which is required to compile *cmdtool*) from MIT; see the Preface for details.

When you start a *cmdtool* process on the command line from an existing *cmdtool* window, the second *cmdtool* inherits the "environment variables" of the first, including the setting of the DISPLAY variable. The program running in the second also begins in the current directory of the first shell. Otherwise, there is no real relationship between the two windows.

The basic operation of *cmdtool* will be familiar to anyone who has used an interactive computer terminal; you should be able to work productively immediately. You type a command, hit return, and see the results. As well, terminal emulators aren't limited to the fixed size of twenty-four lines by eighty characters; you can make them any size.

But *cmdtool* is more than just a "dumb terminal." Two useful features of most modern terminal emulators are an optional scrollbar to review text that has gone "off the top" of the display, and a "cut and paste" facility that lets you rearrange text or even copy it from one terminal emulator window into another. *Cmdtool* includes these, but also has comprehensive text editing features built in.

The *shelltool* and *cmdtool* programs have been part of Sun's graphics offerings almost as long as there have been Sun Workstations. The SunView™ window system used under SunOS Release 3 provided these programs as its standard terminal emulators for several years before the X Window System was even invented. It is not surprising, then, that these two programs offer the same kind of functionality as does the MIT X terminal emulator *xterm* but in different ways.

In those days, you had to remember to use *cmdtool* for full-screen programs such as *vi*, and use *shelltool* for programs that used a simple tty-style interface (there was also *gfxtool* to start up certain types of graphics clients). Under OpenWindows, these two terminal emulators are in fact just two names ("links") for the same program, so for convenience I'll simply use the term *cmdtool* to refer to both of them. The most significant difference between them is that *cmdtool* starts up with scrolling on, and *shelltool* comes up with scrolling off.

The main philosophical difference between *cmdtool* and the MIT terminal emulator *xterm* discussed in Appendix A, *The xterm/olterm Terminal Emulator* is that *cmdtool* tries to let you do full editing on your terminal's history, while *xterm* preserves the history intact and only lets you copy from it, and paste words or lines only into the current input buffer. *Cmdtool* thus carries the notion of "putting shell history into the terminal" one step further than does *xterm*, at a slight cost in efficiency. In fact, *cmdtool* has built into it a complete text editor, which can be used to edit either the shell history or separate text files. This editing facility can also be started up as an editor, called *textedit*; we will discuss this editor here, because it is so similar to *cmdtool*. Another difference is that *xterm* tends to be a little faster than the current version of *cmdtool* at inserting characters, which may make it preferable for touch typists and others who type very rapidly.

5.1 Terminal Emulation and the sun-cmd Terminal Type

If you've used more than one kind of terminal or terminal emulator, you probably realize that they don't all behave the same way. In particular, terminals differ in how they respond to *escape sequences* or series of special characters typed by you or sent from the computer to the terminal. One terminal might clear its screen upon receipt of a Control-X character, for example, while another might need the four-character sequence <ESC>H<ESC>J in order to clear the screen, where <ESC> means the ASCII control character "Escape" or just "ESC". As a result, it can be very hard to write computer software to drive all the different terminals in the world. The computer industry has taken two approaches in trying to tangle with this problem. On the one hand, an ANSI Standards Committee has drawn up a standard, based largely on the DEC VT100 terminal, to specify how terminals should react to escape sequences. And many manufacturers (and writers of terminal emulators) are using this standard.

On the other hand, there are still many terminals in widespread use that do not conform to this standard, and an equal number of variations among those that claim to. As a result, UNIX systems include one of two similar facilities, called "Termcap" and "Terminfo", which are databases of the "capabilities" of all the terminal types, or the "info" needed to drive each type of terminal. Termcap/Terminfo provide information on literally hundreds of different brands of terminals, and are used by screen editors such as *vi* and *emacs* to determine how to drive the particular terminal you are using.

But the trick is that you must somehow tell the software what type of terminal you are on. If you omit this, or get it wrong, command-line commands such as *ls* and *cat* will still work, but screen editors and paginators—any screen-oriented software—will mess up.

How do you tell the system what kind of terminal you have? For each type, there is a name. The name *sun-cmd* is used for the *cmdtool* terminal emulator. The name *xterm* refers to the *xterm* terminal emulator discussed in Appendix A, *The xterm/olterm Terminal Emulator*. Both the *cmdtool/shelltool* and the *xterm* terminal emulators normally set their own TERM environment variable. You should be careful not to override this terminal type in your shell initialization files. For details, please refer to Section 3.5, "Customizing your Session Start-up" on Page 72, and the *cmdtool* reference page in Part Three of this Guide.

Additionally, *cmdtool* interprets a variety of escape sequences of its own. These allow you to resize the window, change the contents of the titlebar, etc. The escape sequences are described in Appendix E, *Control Sequences for xterm and cmdtool*.

5.2 Resizing a cmdtool Window

The X Window System and the OPEN LOOK GUI allow you to change the size of almost any window at almost any time. Most X Window System programs respond reasonably well to having their size changed. However, when a terminal emulator is resized, there are implications for any screen-based clients it is running at the time. The bottom line is that it's OK to resize a *cmdtool* window, but not *while* it is running a full-screen program such as a screen editor or word processor. Most OpenWindows clients *other* than terminal emulators can be resized at any time. A real terminal normally cannot be resized (except with a steam roller), so some screen editors get upset when the terminal is resized. However, several means have been worked out to solve this problem. First, whenever an editor such as *vi* or *emacs* starts up using the *sun-cmd* terminal type, it asks the terminal emulator for its current size, so it doesn't matter whether you have resized it or not. The *resize* command described in the previous chapter is not needed and in fact does not work with terminal emulators other than *xterm*.

However, a problem comes up if you resize a window *while running* a screen editor. Some versions of UNIX allow such a program to be notified (by a signal named SIGWINCH) when its window (terminal) size changes. However, the SunOS 4.1.x version of the *vi* screen editor apparently is derived from the System V *vi* rather than the current Berkeley version, and does not respond to SIGWINCH. Therefore, do not resize a terminal emulator *while* a screen editor is running in a *cmdtool* or *shelltool* window. †

As mentioned in the previous section, you can resize a *cmdtool* dynamically using escape sequences. Example 5-1 is a short shell script that does so.

```
#!/bin/csh -f

# Simplified from a script from Jay Plett jay@silence.princeton.nj.us>
#
switch ( $#argv )
    case 0:                 # no args, set default
        set columns = 132
        set rows = 34
        breaksw
    case 2:                 # two aregs: rows cols eg 24 80
        set columns = $2
        set rows = $1
        breaksw
    default:                # else fall through
        echo Usage: `basename $0` '[rows cols]'
        exit 1
endsw

if ( $term == sun-cmd ) then
    @ cols = $columns + 4
else
    set cols = $columns
endif

stty rows $rows columns $columns# tell unix new term size
```

† The *vi* sequence

```
Q
:set term=sun-cmd
:vi
```

will usually fix it if you accidentally resize a window running *vi*.

```
echo -n "^[[8;$rows;${cols}t"   # tell cmdtool (N.B. "^[" is ESC)
```

Example 5-1. C Shell Script to resize cmdtool

If you install this script in your *bin* directory as *cmd-resize* and mark it executable, you can give the command

% **cmd-resize** *30 90*

to make your current *cmdtool* window be thirty lines deep by ninety columns. If you make the window smaller than its current size, press the *End* key or Ctrl-Return to move to the bottom of the current window.

While we're on the subject of useful scripts for *cmdtool*, here are two more. The first one sets the title, and the second sets the icon label.

```
#!/bin/sh
# cmd-title - set the title in a cmdtool window
echo -n "^[]l$*\\"

#!/bin/sh
# cmd-iconlab - set the icon label in a cmdtool
echo -n "^[]L$*\\"
```

These use the *ucb* version of the *echo* program; if the *-n* appears on your screen when you run them, set */usr/ucb* to the front of the search path in the script.

5.3 The cmdtool Menus

Cmdtool's menu layout is simpler than that of *xterm*; there are two main menus, depending on which mode you are in, and a greater reliance on X resources and on initialization files than is the case with *xterm*. When you start the program in *cmdtool* mode, you get the *cmdtool Term Pane* menu; when you start it in *shelltool* mode (or start it as *cmdtool* but are running a full-screen application such as *vi*), you get the *shelltool Term Pane* Menu.

5.3.1 The Cmdtool Term Pane Menu

The *cmdtool Term Pane* menu has six selections: *History*, *Edit*, *Find*, *Extras*, *File Editor*, and *Scrolling*. Most of these will be described below, in the section on Editing. The *Scrolling* selection is an exclusive choice: on or off. This item's name belies its effect: when you are in *cmdtool* mode, scrolling is on. When you turn scrolling off here, poof! You are now in *shelltool* mode, and the main menu is now the one listed for *shelltool*.

5.3.2 The Shelltool Term Pane Menu

This menu has only four choices: *Enable Page Mode*, *Copy*, *Paste*, and *Enable Scrolling*.

The *Enable Page Mode* causes pausing after every screen full of text, rather like the convenience of having *more* or *pg* built into your terminal so that it benefits every program. However, this mode is disabled if you go into *cmdtool* mode.

The *Copy* and *Paste* selections are the same as those in *cmdtool*'s *Edit* sub-menu, and likewise behave as mentioned below under Editing.

The *Enable Scrolling* item turns on the scrollbar feature. This has the side effect of flipping you back into *cmdtool* mode.

5.4 Using the OPEN LOOK Scrollbar

Many OPEN LOOK programs, including *cmdtool*, allow you to view on the screen one part of a larger "virtual screen." The OPEN LOOK mechanism that controls this is called a *scrollbar*, and allows you to move the on-screen window up or down either one item (such as a line of text) at a time, or smoothly drag it up or down. Unlike most other GUIs, the OPEN LOOK scrollbar also allows some other functions, such as splitting the view of the larger background into several smaller, independent on-screen views.

Note that *cmdtool* normally includes a scrollbar, while *shelltool* does not, unless you have enabled it from the window menu. You can have your OPEN LOOK scrollbars placed on the left side of your applications or the right side, either by using the Workspace Manager (see Chapter 6, *Using the OPEN LOOK Window Manager*) or by specifying an X resource variable (see Chapter 12, *Setting Resources*).

There are two other types of scrollbars you are likely to run into while using The X Window System. The Athena Widget scrollbar is described in Appendix A, *The xterm/olterm Terminal Emulator*. The Motif scrollbar is described in Appendix M, *OPEN LOOK and Motif*. In this section, we will concentrate on the OPEN LOOK scrollbar. Once you've learned it, the others can be learned easily.

First, let's review the parts of the scrollbar, as shown in Figure 5-1Note that it looks much like an elevator riding up and down on an elevator cable. And the position of the elevator within the overall scrollbar shows where you are, proportionally, in the text or list that you are scrolling; if the elevator is at the top anchor, you are seeing the top of the text or list; if it's halfway between the anchors you are seeing the middle, and so on.

The most common use of the scrollbar is by clicking SELECT on the elevator. Pressing and holding SELECT on the center box (the drag area) lets you slide the scrollbar up or down at will. Clicking SELECT on the up arrow will move the elevator up; clicking it on the down arrow will move the elevator down. As the elevator moves up, the part of the text or list that you see moves up; in other words, you move closer to the top of the file. However, if you've never used a computer window system with scrollbars before, don't be thrown off by what happens; when you click on the up arrow, the text on the screen moves down! This is just a consequence of moving the view window upwards, and is analogous to the fact that, looking out the side window of a car driving forward, the scenery appears to move toward the back of the car. This is so obvious that you never think about it, once you've seen it a few times.

Note that when you click on either arrow, the pointer jumps† when the elevator moves, to stay on the arrow. Thus you can click SELECT repeatedly to move several lines at a time without having to move the pointer each time. In fact, if you hold down SELECT you will

† You can disable this in the *Properties* editor.

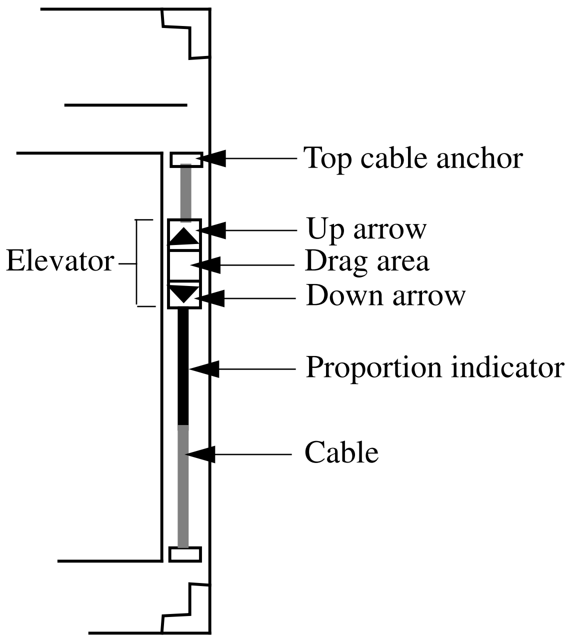

Figure 5-1. Parts of the OPEN LOOK scrollbar

notice that it "auto-repeats" like the keys on your keyboard: it keeps moving along until you reach the end of the data.

You can also click on either cable anchor to return the elevator to that end. Clicking on the top cable anchor moves you to the top of the file; clicking on the bottom anchor moves you to the bottom.

What's in between the elevator and the cable anchors? The cable, of course. You can click SELECT on any point of the cable. When you do, the elevator moves up or down by one screen full of text each time and again, the pointer jumps if necessary.

You may have noticed that one portion of the cable is thicker than the rest of it, or rather is black while the rest is gray. The black part is the so-called *proportion indicator* that shows how big the on-screen portion of data is in *proportion* to the total amount of data viewable. The proportion indicator is always attached to the elevator. If the elevator is at the top of the column, the proportion indicator will be below it; if at the bottom, above it. If the elevator is somewhere in the middle, the proportion indicator will be partly above and partly below, in proportion to where you are in the total data. If the data is so huge that the proportion indicator would be invisible beneath the elevator, a tiny proportion indicator is used. If all the data is visible, as when you start up a *cmdtool* program, the entire cable is black.

5

5.4.1 Jumping with the scrollbar menu

The scrollbar has its own menu, which you can get at any time by moving the pointer into the scrollbar and pressing the MENU button. There are usually four items on it: Here to top; Top to here; Previous, and Split View. These simply allow you to move the top line that is on the screen ("top") to where the pointer was pressed ("here"), to move the lines where the pointer was pressed to the top of the screen, and to undo either of those two move operations (the fourth item, *Split Views*, is optional, at the discretion of the application writer; we'll discuss it later). For example, suppose that we have this window:

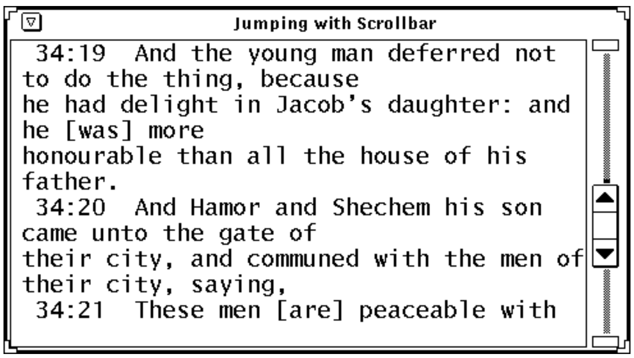

Figure 5-2. Moving top to here: before

and want to scroll it so that the first line ("34:19 And the young man...") is near the bottom. We just move the pointer into the scrollbar near the bottom, click MENU, and release on *Top to here* to move the top lines to where we are, as shown in Figure 5-3

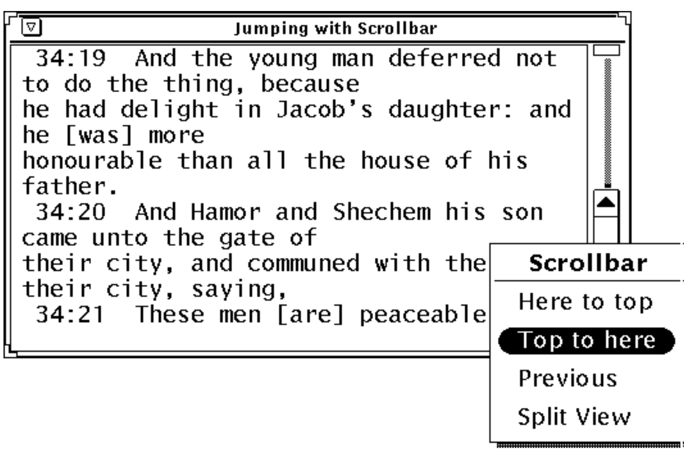

Figure 5-3. Moving top to here: before

After we release on *Top to here*, the top line will be at the bottom, as in Figure 5-4

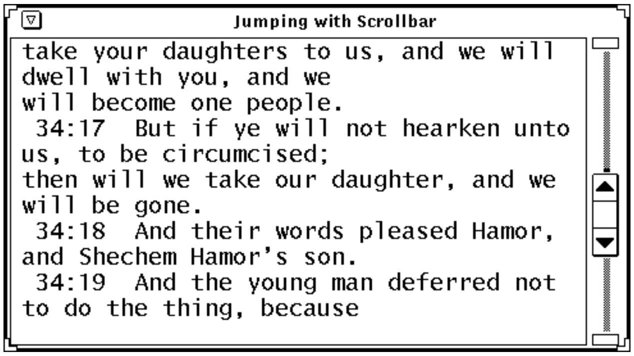

Figure 5-4. Moving top to here: after

5.4.2 Splitting with the scrollbar

A unique feature of the OPEN LOOK scrollbar is that it can be used to "split" the view in most applications. You often want to see two (or even more) parts of a file, and possibly copy text from one place to another. In the next section we'll see how to copy text. Here we'll learn how to get multiple views of the *cmdtool* history. You can split views either by using the *Scrollbar* menu item *Split Views* or by using the cable anchors. Let's say we have a *cmdtool* window that we've been typing commands in for some time; it may look like Figure 5-5:

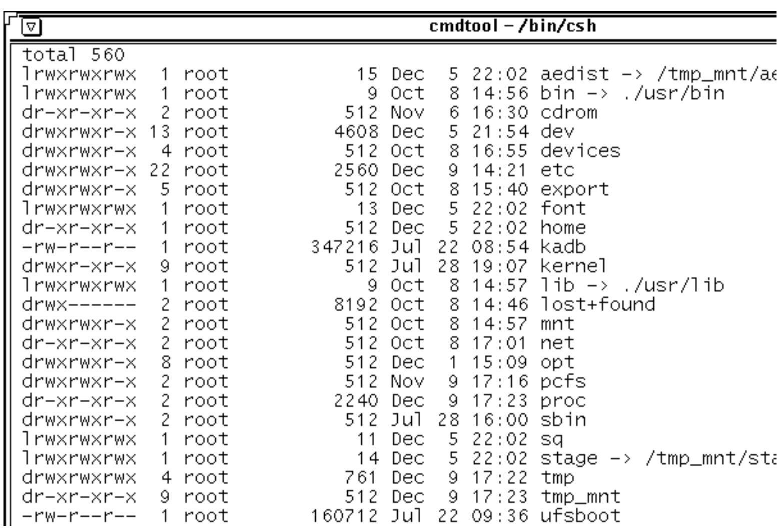

Figure 5-5. Cmdtool window before splitting

To split the window into two views, move the pointer into the scrollbar and press the MENU button, and you will see the menu in Figure 5-6.When you release the MENU button on *Split Views*, you will have two views, each with its own scrollbar, as shown in Figure 5-7

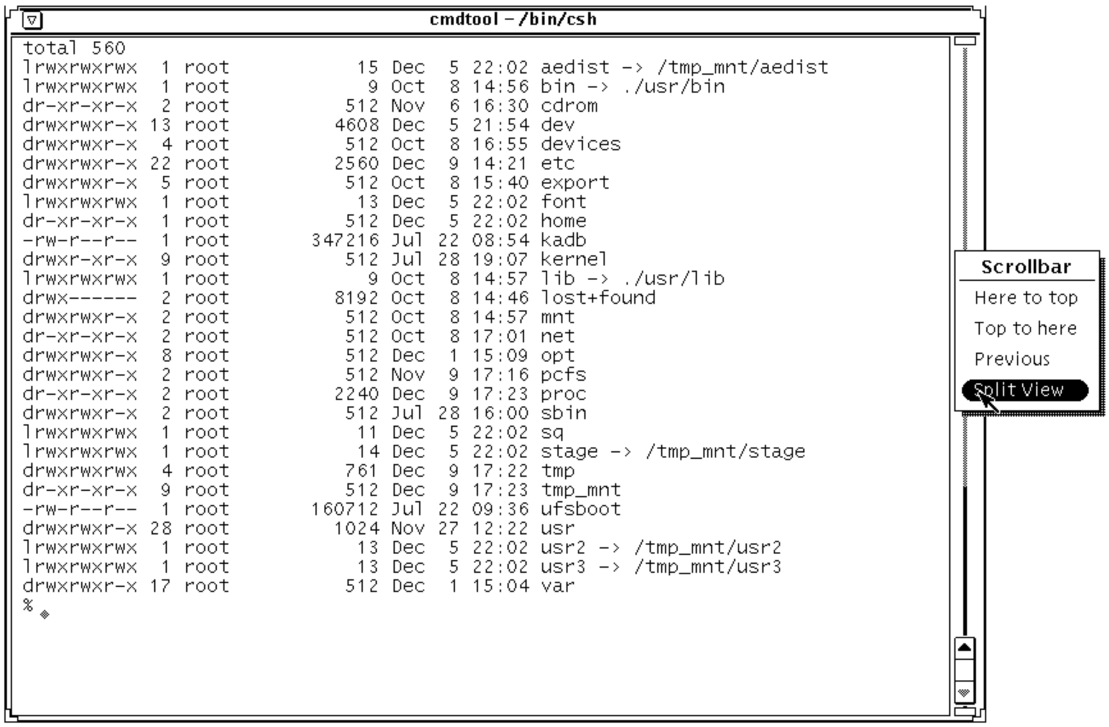

Figure 5-6. Cmdtool window: scrollbar splitting

Notice their initial positions; the top one is left where it was, and the new one begins at the top of the file. Furthermore, their relative sizes reflect where you pulled down the menu; to get two views of equal size, click the MENU button near the middle of the scrollbar. As an interesting experiment, to prove to yourself that these are but multiple views on a single shell history, move both scrollbar elevators to their bottom anchors, and type a new command such as *date* and run it. You should see the command, and its output, appear simultaneously in both views, as shown in Figure 5-8.

You can also split a view by dragging the cable anchor. For example, click and hold SELECT on the top cable anchor, and drag it down: notice that a thin horizontal line follows it, to show the position more clearly. Drag the anchor and line to the place where you want the split to be, and let go. Now you have two views into the data.

If you decide you no longer need multiple views, you can join them again. When you have split views, the scrollbar menu gains an additional last item, *Join Views*; if you release here, the views will be joined. You can also join views by dragging one cable anchor of a view on top of the other cable anchor in that view; the view will be joined with the other view.

Finally, you can have as many split views as you need. Two, three or four split views are not uncommon. An application that uses horizontal scrollbars, such as for viewing a large bitmap or for viewing long lines of text, can allow you to split the view into several vertical views as well. It is up to the application program whether you can split a given scrollbar or

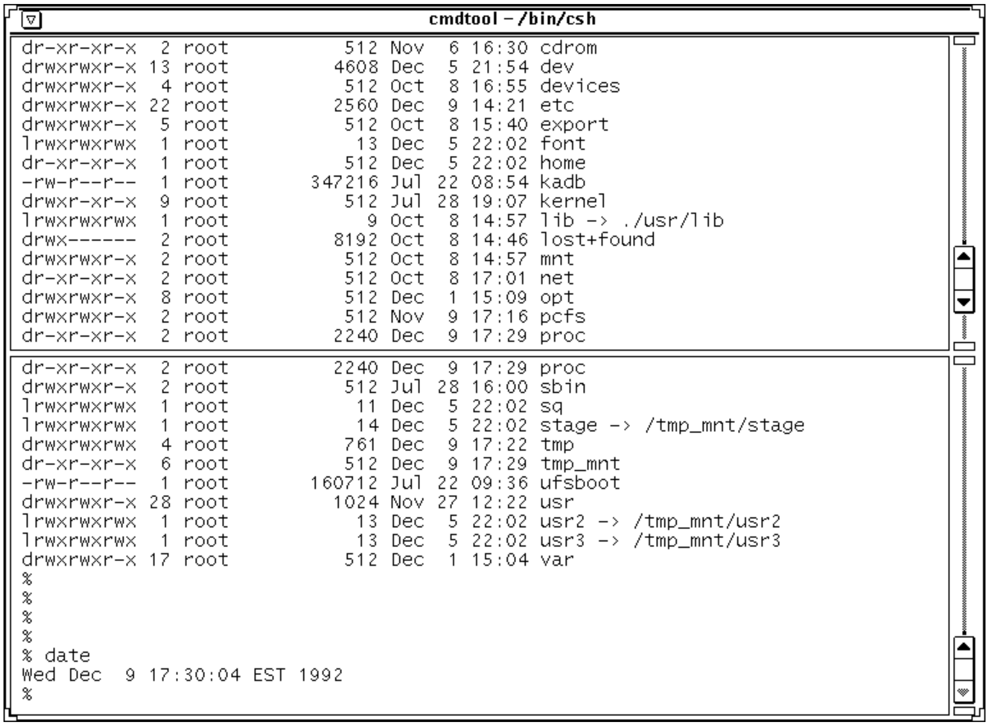

Figure 5-7. Cmdtool window: after splitting, two views

not. If a given scrollbar isn't splittable, it won't have *Split View* in the scrollbar menu. Everything that we have said here about scrollbars applies to scrollbars used in *any* OPEN LOOK application, not just *cmdtool*.

5.5 Copying and Pasting Text Selections

A powerful capability of most OPEN LOOK text application programs is the ability to copy and paste portions of text. You can select a few letters, a word, several words, a whole line, or even several lines. Once selected, text can be copied into an internal buffer and then *pasted* or copied into the input once or as often as you wish. This selection mechanism is similar to, but not identical with, that provided by the MIT X versions of *xterm* and related programs. First we'll see how OPEN LOOK does it, then we'll look at cutting and pasting between OPEN LOOK applications and MIT-style programs.

5.5.1 Selecting Text to Copy

To select some text for copying, you must first move the pointer to the start of the text. Then, you press SELECT, and can either drag the pointer ("wipe") across the text or mark your selection using the pointer buttons.

To wipe across the text, just hold the SELECT key down and move the pointer. As you go, you'll see that the text is highlighted by displaying it in reverse video, as in Figure 5-9[†]

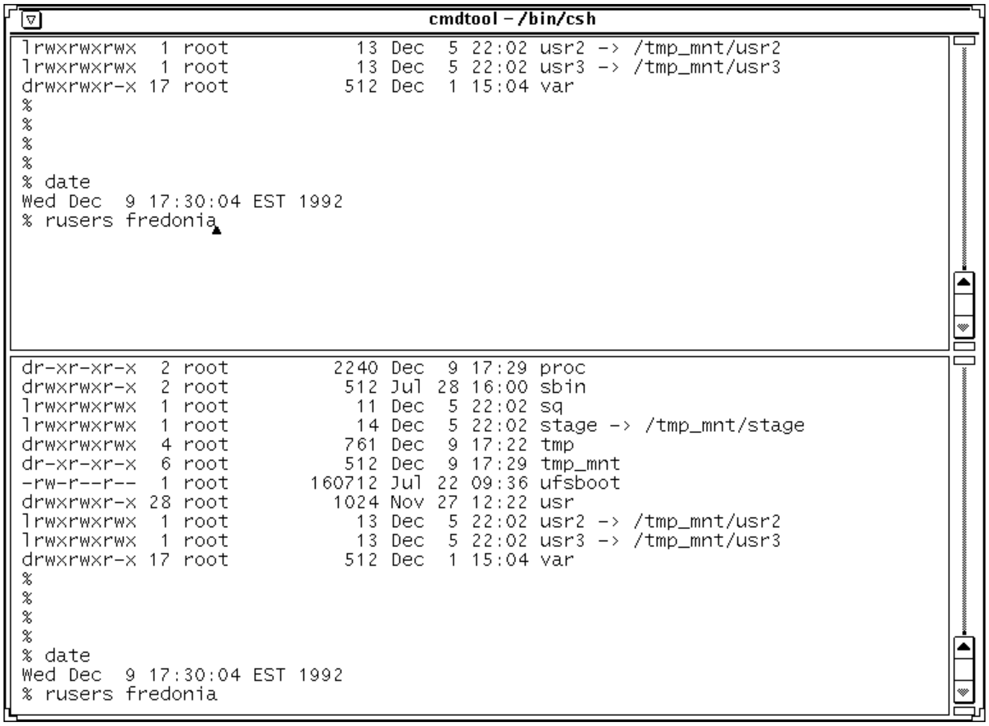

Figure 5-8. Both views reflect a single reality

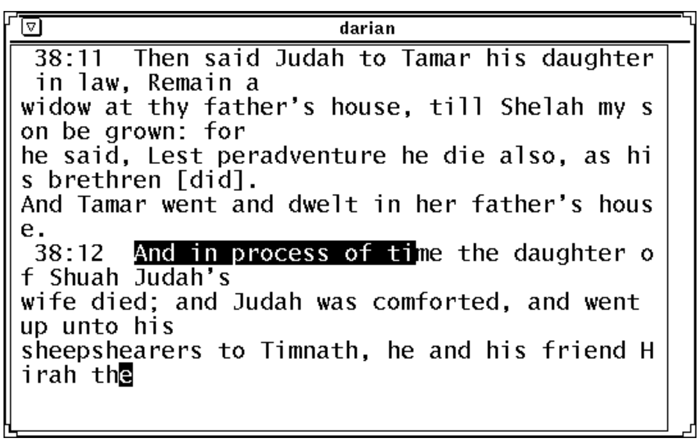

Figure 5-9. Selecting Text by "sweeping"

† Remember that this reverse video is different from the single-character *vi* cursor, if you are using that editor. *Vi*'s cursor shows where it thinks the action is; the selection highlighting shows what text has been selected for later re-use.

You can let go at any time, and that is where the selection will stop. As usual, you can lengthen or shorten the selection with the ADJUST (middle button). With a bit of practice, you can lengthen or shorten it by as little as one character at a time.

Another method of making a selection is by multiple clicking. Clicking and releasing the SELECT pointer button once sets the point at which the selection will start, as before. But clicking and releasing it again very quickly (a *double click*) will cause the selection to include the entire word around the selection starting point:

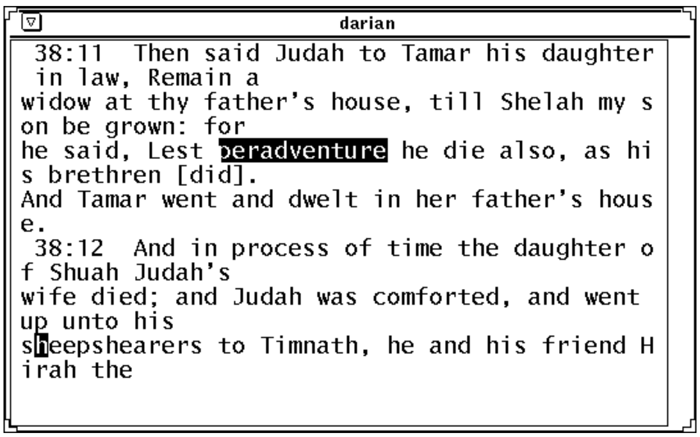

Figure 5-10. Selecting a word by double-clicking

You can also exercise your fingers in a blaze of clicking: three quick clicks in a row, or a *triple click",* will select the entire line on which the pointer is. And if you are really quick, you can *quadruple click* or type four quick clicks; this selects *all* the text that is visible in the window.†

Now you have some text selected, and you can act on it.

5.5.2 Copying or Cutting the text

Once you have some text selected, you can move or copy the selected text into an internal text buffer. If you move it, the original copy is removed from the screen and put into the selection buffer, and this is called a *cut* operation. If you copy it, the original text is left on the screen as well as being put into the selection buffer; this is called a *copy* operation. The *cut* operation is only available while scrolling is on, that is, in *cmdtool* mode but not running a full-screen editor or other application. Again, note that in a *cmdtool* window, this cutting and copying is independent of any cut-and-paste logic in the program you are running, such as *vi*'s Yank and Paste commands or *emacs*' kill-text and yank operations. Also, cutting (as opposed to copying) is automatically disabled when running screen editors *vi*, *emacs*, etc.) or any other program that takes over control of the emulated terminal's screen.

† The Properties editor *props* lets you configure a maximum interval at which a pair of mouse clicks is treated as a double-click.

To copy text that has been selected, you can use the application menu item *Copy*. In *cmdtool* mode this is found under the *Edit* item; in *shelltool* mode, it is on the main menu. Because these are common operations, however, there are normally keyboard shortcuts. On a Sun keyboard, for example, the L6 key is a shortcut for the *copy* operation, L8 is short for *paste*, and L10 is short for *cut.*

5.5.3 Replacing the text

As a shortcut, you can *replace* selected text from the keyboard just by typing into the window or text field that contains the selected text. This in effect combines a *cut* operation with an insert, all in one move.

5.5.4 Pasting Text Selections

A common operation is to copy a section of text from one *cmdtool* window into another, or into a different place in the same window. If you wish to re-use a chunk of text that has been copied (or cut), use the *Paste* operation to insert it. If you want to insert it into text that you are editing with *vi* or *emacs*, for example, move that editor's cursor to the place you want it inserted, make sure the editor is ready to receive it (in the case of *vi*, for example, you must be in insert mode), and choose *Paste* from the application menu; it's found in the same place as *Copy* was found above. The keyboard shortcut on a Sun-3 keyboard is L8. For example, if we wish to paste a longer selection of the Biblical text from above into this *vi* window:

Figure 5-11. Vi window before pasting

we need only make sure that *vi*'s cursor is at the right place, and type (for example) an "o" command to Open a new line and enter Insert mode, and we can paste the text. We can either pull down the window menu and release on Paste. This is shown in Figure 5-12

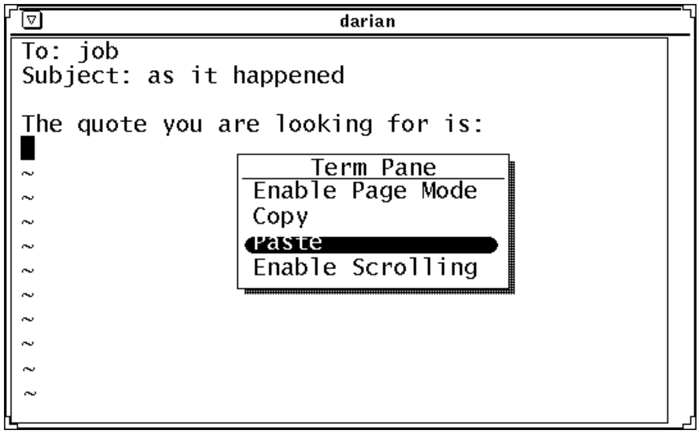

Figure 5-12. window during pasting

Or we could type the "o" command then hit the Paste key (L8). In any event, the text will be pasted in, as shown in Figure 5-13. You will wish to disable the *showmatch* and *autoindent* options in *vi* if you are using them, to avoid unwanted interactions.

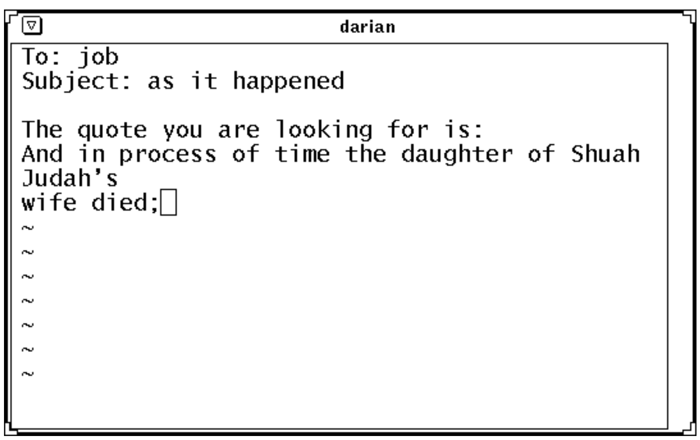

Figure 5-13. Vi window after pasting

If you wish to run the selected text as (part of) a command, be sure you are at a shell prompt in your *cmdtool* window, and paste the text in. For example, here we are about to copy and paste a job name that showed up in the output of a *uustat* command, to re-use it as the argument of another. The syntax

uustat -k ***jobname***

is used to kill a previously submitted *uucp* (UNIX-to-UNIX copy program) job. This is a good example, since the names that *uustat* generates on this version of *uucp* tend to contain case changes and letters and numbers mixed together, so they are tedious to retype. After typing the part of the command that doesn't include the jobname (i.e., the "uustat -k"), we select the one word of text we want ("sqN7e04") just by double-clicking on it:

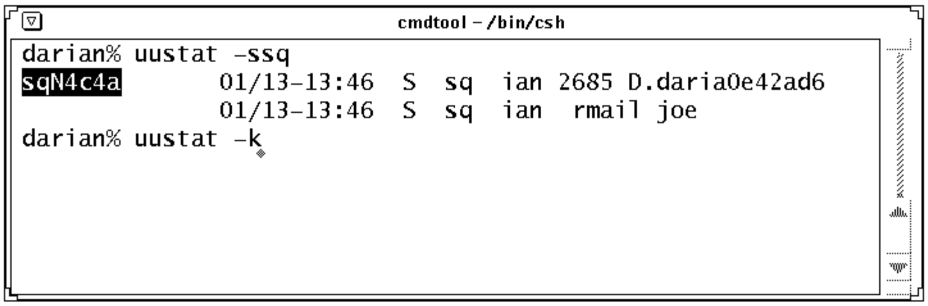

Figure 5-14. Re-using part of a shell command: before

Then you have two choices:

- press the keyboard keys for COPY (L6) and PASTE (L8) to insert the text.
- Click and hold SELECT pointer button, and drag a copy of the selection to the insertion point (as shown in Figure 5-15), then release the pointer button

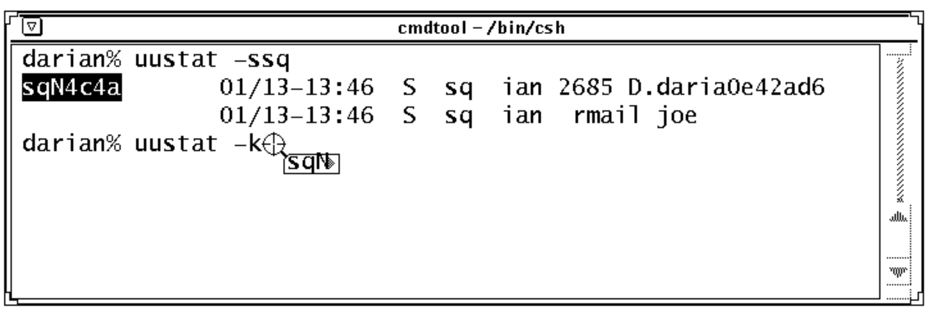

Figure 5-15. Dragging the text with the cursor

When you've copied the text, you may need to press the RETURN key to run the command. The resulting screen should look like Figure 5-16.

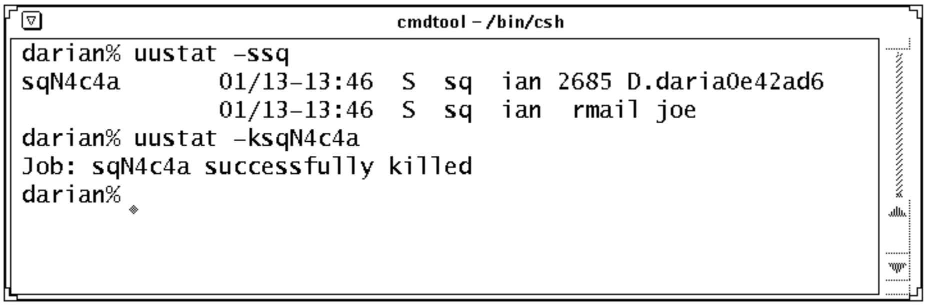

Figure 5-16. Re-using part of a shell command: after

If you have picked up some text from earlier in the history log by using the scrollbar to move backwards, you have to get back to the insertion point at the very end of the history (the current or active line). You can just click SELECT on the bottom scrollbar anchor to move the scroll to the bottom line (this does not de-select any highlighted text you may have), then click on the last line (after the shell prompt). This is a common operation, so Control-Return (press RETURN while holding the Control key) is a shortcut for this, as is the End (R13) key on a Sun Type 4 keyboard. (By contrast, *xterm* moves to the end of the last line automatically when you type anything, even a space; this is a bit faster, but is a consequence of *xterm*'s read-only history log.)

If you want to copy text from one OPEN LOOK window to another, here is the sequence of steps:

- In the source window, select the text;
- Click and hold the SELECT pointer button, and drag the text to where you want it. Note that as in Figure 5-15, the cursor becomes a "gunsight" when you are over a drop target.
- Release the pointer button, and the selection will be copied.

There are some occasions when you cannot use this mechanism. A common case is the *rlogin* and *script* programs, which put the terminal into so-called *raw mode*, in which drag-and-drop does not work (more generally, command-line editing with the mouse does not work). In this case, use the more explicitly copy-and-paste method:

- In the source window, select the text;
- Copy the text into the paste buffer (use *Copy* menu item or *Copy* keyboard button, L6);
- Move the pointer to the new window, and if using click-to-type focus, click SELECT in the target window;
- If using an editor in the target window, make sure the editor is in insert mode at the appropriate point in your file. Alternately, if you want to paste the text into an OPEN LOOK Text Field, click SELECT on that text field to move the insertion caret there.
- Paste the text (use *Paste* menu item or *Paste* keyboard button, L10).

Finally, there is one even shorter method:

- Click SELECT to get an insertion caret where you want the text, if necessary
- Press *and hold* the Paste key (L8)
- Select the text you want copied using the SELECT pointer button. It should be underlined, rather than highlighted in reverse video.
- Let go of the Paste button, and the text will be copied.

To *move* text from one window to another rather than *copying* it, you would follow the same steps for any of these methods except use *Cut* instead of *Copy.*

It sounds like more work than it is. Try it a few times and you'll see how easy it is. For anything with tricky typing, or anything longer than a word or two, it is quite a bit faster (not to mention more accurate) to copy it rather than to re-type it. Once text is in the computer correctly, *let the computer do the work*.

5.5.5 Editing and Saving the History Log

The *cmdtool* program has the ability to save its History Log into a file, using the main menu's *History* menu. This can be used, for example, to document how a program actually behaves, for purposes of writing technical documentation or for filing a bug report. You can edit the *cmdtool*'s history log prior to saving it. Editing can be used, for example, to remove errors that you made, such as running some command with an option missing, without having to flip into a text editor.

5.5.5.1 Clearing the log - a clean start

If you have been using your *cmdtool* session for a while before starting to make a record of some program's behavior, you will probably want to start with a clean slate. Pull down the main menu, then pull right on *History* and further right onto *Clear Log*. When you release this item, the history log will be cleared out, and you can start typing your example.

5.5.5.2 Other Editing

There are many other editing options, described in the following section. These can be used in many places where OPEN LOOK applications allow editing of text fields or strings.

5.5.5.3 Saving

Once you have the log in the form you want it, you can save it to disk using the *History* menu item *Store log as new file*. Just release on this and you will be prompted for the name of a file to save the history log into. Once you type the name and click on the *Save* button, the history log will be written into a file.

5.6 Editing Text in OPEN LOOK Applications

OPEN LOOK specifies the methods used for editing from the mouse and keyboard. This is not to prevent use of more sophisticated editors, such as *vi* and *emacs*, but only to provide a standard set of editing functions that can be used anywhere in the X and OPEN LOOK environments that text must be edited from the mouse and keyboard. This discussion of editing the history log applies to any OPEN LOOK application that uses text or text fields, such as the OPEN LOOK-specific text editor program, *textedit*, as well as to *cmdtool*'s history log. The OPEN LOOK text editing facilities give you a comprehensive text editor, including cut/copy and paste, search and replace, and the ability to filter text through a series of external programs.

Cutting text out has been described above: select the text using the mouse, and cut it using either the keyboard shortcut (L10) or the *Cut* main menu item.

You can insert text at any point. Position the pointer where you want to begin inserting, and click SELECT. You will see a small insertion caret, and can begin typing at once. Continue typing until you have said your piece. If you make a mistake, you can correct it with the backspace key (some applications also let you use the DELETE key). Most other keys, including control characters, will be inserted directly.

The main menu, called *Term Pane* in *cmdtool* and *Text Pane* in *textedit*, has an *Edit* item that invokes a series of editing functions, most of them familiar. The bottom three are *Copy, Paste*, and *Cut,* which have been described earlier in this chapter. The others are *Again* which repeats the previous editing change, and *Undo. Undo* can undo either the single most recent editing change, or the entire set of editing changes. The order of these change operations, *Again*, *Undo*, *Copy*, *Paste*, and *Cut* may seem arbitrary, but it is the same order as the keyboard shortcuts for these actions on the Sun-3 keyboard: Again is L2, Undo is L4, Copy is L6, Paste is L8 and Cut is L10.

The next item in *cmdtool*'s *Term Pane* and *textedit*'s *Text Pane* is *Find*, which lets you locate and optionally replace text. It leads to a menu of four items: *Find & Replace...*, *Find Selection*, *Find Marked Text*, and *Replace Field*. The first, *Find & Replace...*, pops up a window with two text fields, one for the text to be found and the other for the text to be replaced. There are several push buttons on this pop-up window:

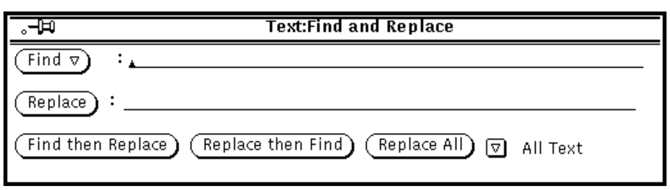

Figure 5-17. Find and Replace Pop-up Window

The *Find* button lets you search forward or backwards after you have entered your text. If you wish to replace the found text, you can select *Replace then Find* to do the replacement and find the next occurrence. Or you can select *Replace All* which will do the replacement in the whole file. The abbreviated menu button at the right lets you select between *All Text* (replace throughout the entire document) or *Here to end* which will only make the change from the current point or found string up to the end of the document. The *Find Selection* item lets you select some text and find it, without having to copy it or retype it in the *Find* text field. The next matching text will be found, if there is one, and left selected, so you can cut or copy it from the new location. This is a common operation, so there is a keyboard shortcut for it. On a Sun keyboard, the L9 (FIND) key will find the selected string if you are in an edit window. Shift-L9 will find backwards. On any keyboard, Meta-F and Meta-

Shift-F will perform the same find operations. If there is no string currently selected, it will try to find the most recently *Cut* or *Copied* string.

Find marked text and *Replace Field* are advanced topics and will not be described here.

The next item in *cmdtool*'s *Term Pane* and *textedit*'s *Text Pane* is *Extras*, a list of odds and ends that runs some external filters on the selected text. The *Format* item runs the *fmt* text alignment program; this is useful for making long lines of text fit, or for joining words into a single line to run as a command. For example, to list (*ls*) the three files highlighted in Figure 5-18,

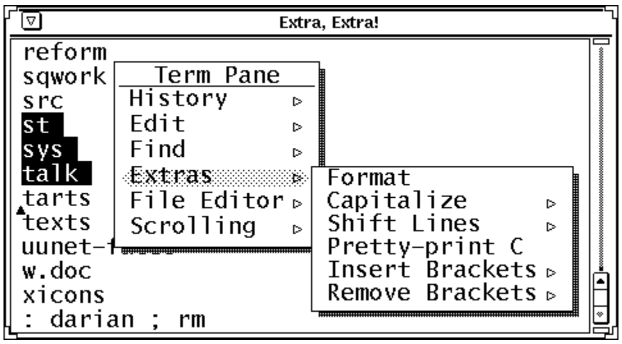

Figure 5-18. Three filenames selected for "formatting"

we select them with the pointer, pull down the *Term Pane* to *Extras*, and release on *Format*. This replaces the three lines shown, with one line:

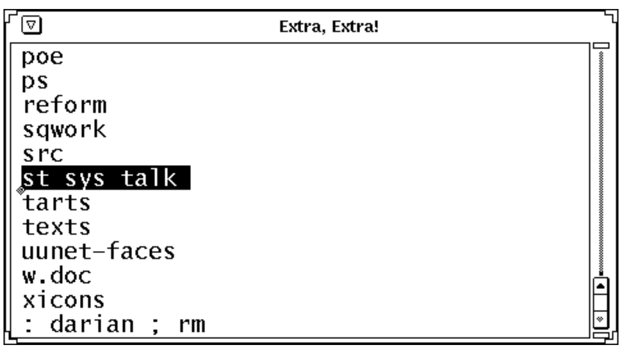

Figure 5-19. Lines joined together

We then highlight this line (either by wiping across it or by triple-clicking), press the COPY key, move to after the *rm* command (either by clicking SELECT to the right of it, or by using Control-RETURN), press the PASTE key, and it's done:

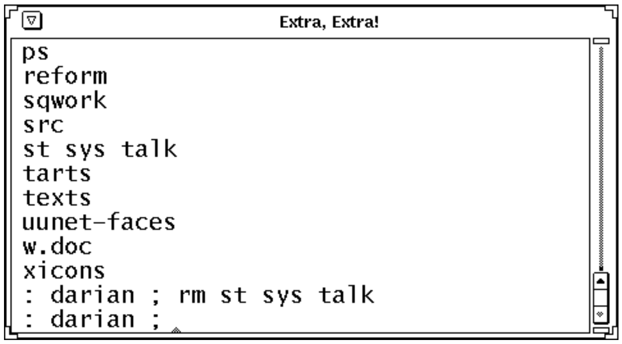

Figure 5-20. The command is done

Admittedly cutting and pasting is not much faster for three short filenames, but for half a dozen long or complicated filenames it can be a significant time-saver.

The remaining items in the *Extras* menu allow you to

- shift the case of text from upper to lower (useful with MS-DOS filenames), lower to upper, or even to capitalize words (useful with people's names);
- shift entire lines to the left or right;
- have C source code reformatted or "pretty-printed", and
- insert and remove various kinds of parentheses or brackets (useful in expressions in most computer programming languages.

These are mostly useful to programmers who already know why these tools are necessary, so they're not explained here. We will show you how to change the functions in this list in **Snuff this dead chapter reference ASAP!**.

The final item in *cmdtool*'s *Term Pane* is *File Editor*. This selection turns the bottom half (or more) of the *cmdtool* screen into a *textedit* window, which behaves just like *textedit*, as described in the next section.

5.7 Editing with Textedit

Textedit is the OpenWindows text editor program. It can be started from the *Workspace* menu, from the OPEN LOOK File Manager, from within a *cmdtool* window, or any place else you need editing abilities. *Textedit* is easier to learn than conventional UNIX text editors like *vi* and *emacs*, but an adept user of one of those editors can get work done more quickly than a user of *textedit*.

All of its editing facilities are the same as those of *cmdtool*, since it contains the same editing program. All you need to learn now is how to change text in the on-screen buffer, and how read and write files plus a few other, less common operations and you'll be a *textedit* user.

5.7.1 The Editing Keys

There are several keys that can be used to edit text in an OPEN LOOK application such as *cmdtool*. These are similar to those in other keyboard editors, most notably to the *emacs* family of editors. But they are not exactly the same as *emacs*, so users of that editor should not get complacent, and users of other editors should not feel disadvantaged.

Before describing the individual keys, we need to explain the notion of *insertion caret* or *insertion point*. There is always one insertion point in the window of an OPEN LOOK application that has any text fields or text windows. This is the point at which most editing operations take place. For example, text that you type is inserted at the caret, while deleting the start of a line deletes from the caret leftward to the start of the line. You can move the insertion point anywhere in the text window just by moving the pointer to where you want the caret, and clicking SELECT. A small caret symbol (♦) will be displayed there.

Several control keys (hold the Control key and type the given letter) have effect on the edit. These are similar to the MIT *xedit* program, however, the list of keys that work here is smaller than with *xedit*. Several keys that work in *xedit* but not here are listed as "No effect" so that you can compare them if you have previously used *xedit*. Table 5-1 lists the control keys and their actions.

Table 5-1. TextEdit Editing Characters

Key	SHIFT reverses?	Function (mnemonic)
Control-A	yes	Move to the beginning of the current line (previous line if repeated).
Control-B	yes	Move backward one character ("backward").
Control-D	no	No effect; inserts a Control-D.
Control-E	yes	Move to the end of the current line.
Control-F	yes	Move forward one character ("forward").
Control-H	yes	No effect; inserts a Control-H.
Backspace	yes	Deletes char to left of cursor
Return	no	Break line at caret.
Control-Return	yes	Moves to end of document
Control-I	n/a	Inserts TAB character.
Control-K	n/a	No effect; inserts itself.

Table 5-1. TextEdit Editing Characters

Key	SHIFT reverses?	Function (mnemonic)
Control-L	n/a	No effect; inserts itself;
Control-N	yes	Move down to the next line. (“next”)
Control-O	n/a	No effect; inserts itself.
Control-P	yes	Move up to the previous line. (“previous”)
Control-U	yes	Delete from the start of line to the insertion point (common *stty* setting)
Control-V	n/a	No effect; inserts itself
Control-W	yes	Delete the word to the left of the caret. (common *stty* setting)
Control-Y	n/a	No effect; inserts itself.
Control-,	no	Word motion to left (under < key).
Control-.	no	Word motion to right (under > key).
Escape key	n/a	No effect; inserts itself.
Delete key	yes	Delete the character to the left of the caret.

5.7.2 The textedit menus

When started from the *Workspace* menu or the command line with no filename argument given, *textedit* looks like this:

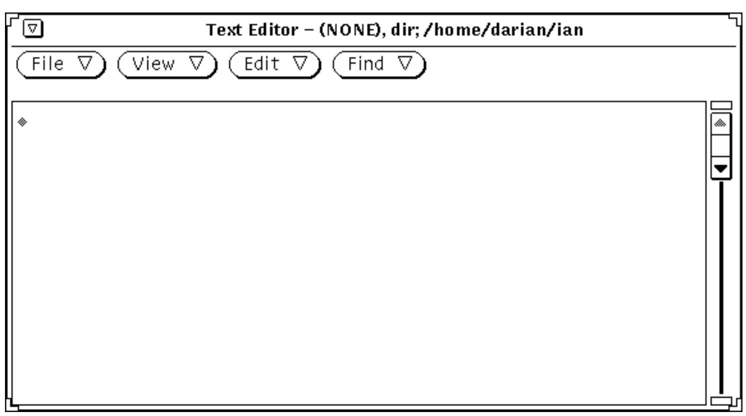

Figure 5-21. Textedit initial screen

The "NONE" in the titlebar simply means that no file has been selected. When enabled within a *cmdtool*, the text editor looks similar except that it only occupies the bottom part of the window; the bottom of the command log is displayed in the top window, like so:

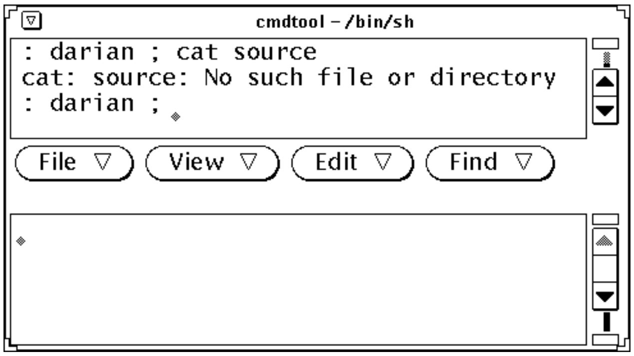

Figure 5-22. TextEdit started from within cmdtool

In either case, you are presented with an empty document. If started with a filename on the command line, of cours4e, *textedit* starts up with the first screenful of text displayed. To create a new document, just begin typing. Or you can load an existing document using the *File* menu. Here is an example of using the editor within *cmdtool* to create a new, short file:

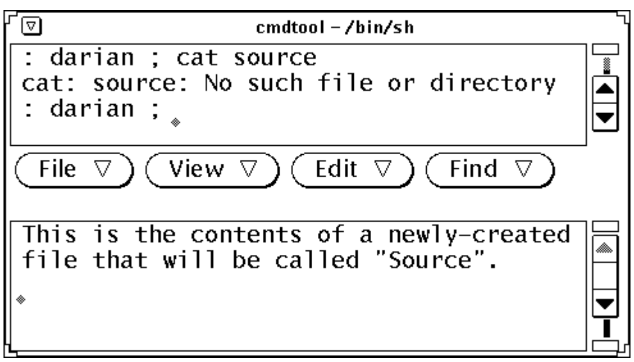

Figure 5-23. Creating a short file with textedit.

Having created the two lines of text, you would save it as a new file, as described below using the *Edit* Hint: you can just click SELECT on the *File* button and the program will do the right thing: save an existing file, or ask you for a name to save a new file under.

The menu bar has the conventional *File*, *View*, *Edit* and *Find* pulldown menus. Each of these pulls down a menu. In addition, these are the same names and in the same order as the first four controls in the *Window Menu*, which also has *Extras*. But *Edit*, *Find*, and *Extras* are the same in *textedit* as in *cmdtool* above, so there is nothing new to say about them. Instead, let us discuss *File* and *View*.

5.7.2.1 The File menu

The *File...* menu, whether accessed from the menu bar or the window menu, has these choices in it:

Load File...
Save Current File
Store as New File...
Include File...
Empty Document

Load File... is used to load a text file into the editor to start editing it. You get a filename dialog window, either a directory browser or a simple dialog with text fields for the directory and filename, and a *Load File* button. The directory text field is initialized to the current directory, so if the file is in the same directory, just type its name and click the *Load* button. If the file is in some other directory, then you can either edit the directory name and type the filename, or just type the full path in the *Filename* field. In either case, click *Load* to actually load the file. And as a shortcut, hitting RETURN after typing the filename will load it, just as if you had clicked on the button. Many OpenWindows programs use this technique: notice that the *Load* button has a double circle or "default ring" to indicate that it is the default action. If you have popped up this dialog but want to cancel the action, you can pull down the window menu of the popup window and select *Dismiss*, which will cancel the Load operation.

When there is nothing in the edit buffer, the default item on the *File* menu is *Load file...* Once you have started editing, the default item changes to *Save Current File*. Clicking SELECT on a menu button performs the menu's current default action. Also, if you try to load a file when you have unsaved text in the text buffer, you will be prompted to save it before it is cleared.

Save Current File saves the contents of the *textedit* buffer back into the file you edited it from.

Store as New File lets you write the changed file in the edit buffer out to a different file than that from which you edited it. This would also be used in the case where you are creating a new file with *textedit*. This popup dialog behaves the same as the one for *Load File*

The *Include File* control is used to insert another file in the place where the insertion caret is, just as if you had typed the file in anew. It prompts with a standard File popup dialog, as above. When you click on the *Include* button, the file is read into your edit buffer, right where the insertion caret is.

A valuable short form for *Include File* is <Meta>-i (hold Meta key and type "i"). If a filename is selected anywhere on the display, it will be pre-loaded into the *Include* dialog, and you just have to hit return there (or click on the *Include* button) to insert the file. If the selected filename is in the current *textedit* window, it will be *replaced* by the contents of the file.

The *Empty Document* control does what it says: it empties the document, or gives you an empty document. If you have unsaved text in the buffer, you are prompted with a Notice.

A successful *Empty* operation forgets the remembered filename, too, so you don't have to worry about accidentally saving an empty file over top of one you were editing.

5.7.2.2 The View menu

The *View* menu has these controls:

Select Line At Number...
What Line number?
Show Caret at Top
Change Line Wrap

Select Line At Number... simply allows you to jump to a given line by number, and select all the text on that line for copying. If you want to get to line 450 of the file you are editing, for example, select this control and you will pop up a small dialog box into which you can type the number 450. When you press the *Select* button, your view of the file will be advanced to line 450, and the text of line 450 will be selected so that you can Cut or Copy it for later pasting:

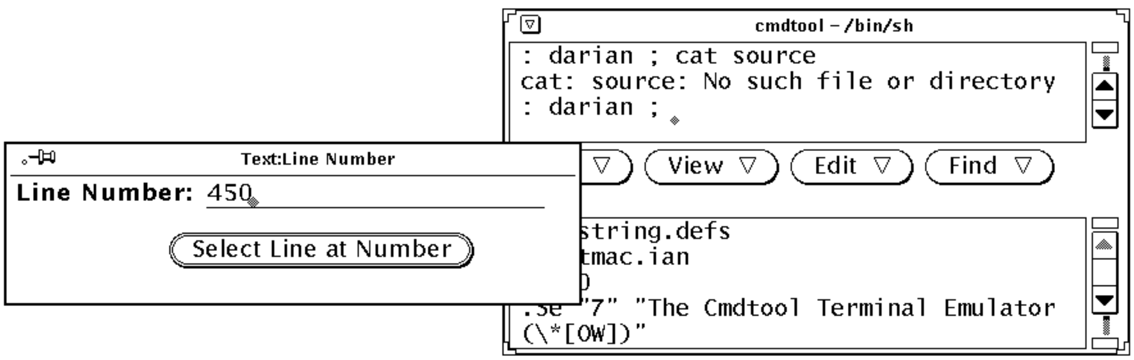

Figure 5-24. Selecting line 450

When we click the *Select* button, line 450 will be highlighted:

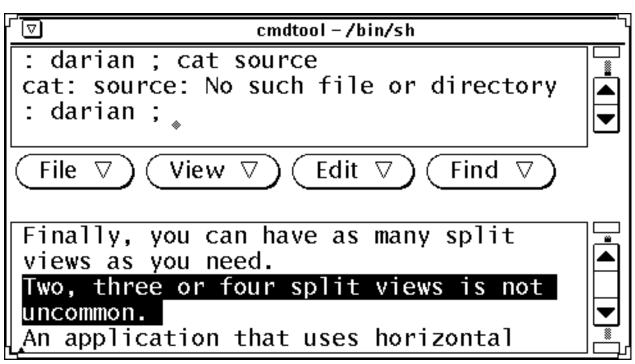

Figure 5-25. Line 450 now Selected

A sort of inverse operation is *What Line number?*, which tells you the line number that the currently-selected text begins on. Let us suppose that we have selected this text from a larger file, and want to see where it is in the file:

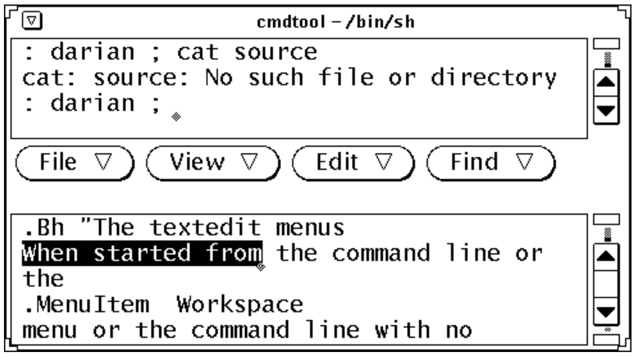

Figure 5-26. What line is this text on?

We pull down the *View* menu and select *What Line Number?* Immediately, this Notice pops up:

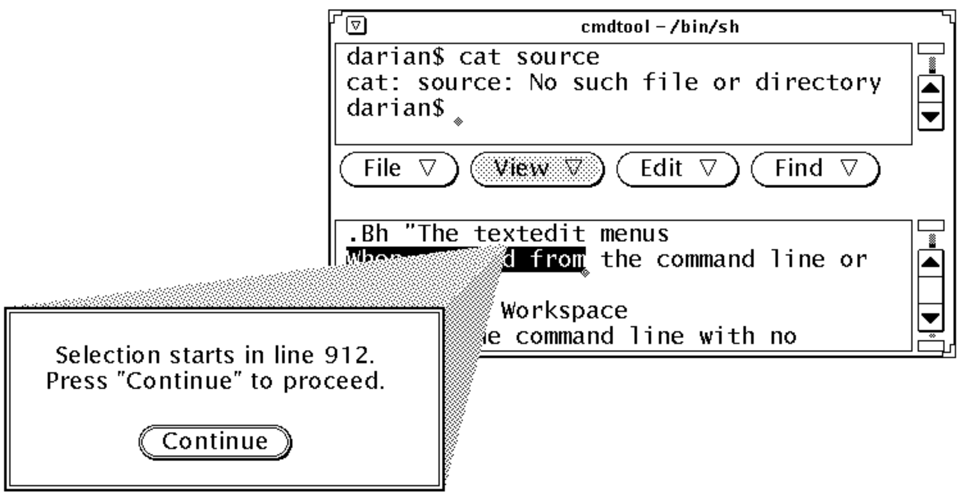

Figure 5-27. It is on line number 912.

and we see that is line number 912 in the file that we are editing.

The menu item *Show Caret at Top* simply moves the text so that the line with the insertion caret is at or near the top of the screen, so you can see the context around it better. You could do this yourself with the scrollbar, so I'm not sure why you'd want it.

The final item, *Change Line Wrap*, lets you control how lines longer than the width of the text window are to be displayed. You can choose from *Wrap at Word*, *Wrap at Character*, or *Clip lines*. The first two are reasonably self-explanatory; text that won't fit on a line is wrapped either at a word boundary or at a character boundary (i.e., right at the right margin). The last item doesn't actually truncate the lines in the file, but only ignores them for purposes of the display. In fact, none of these options actually changes the text; they only affect the way it is displayed. If you find yourself setting this each time you start an edit window, you may wish to customize it, as described in **Snuff this dead chapter reference ASAP!**.

5.7.2.3 The Edit, Find, and Extras Menus

As mentioned, the *Edit*, *Find*, and *Extras* in *textedit* have the same controls and function in exactly the same fashion as they do when editing a history item in *cmdtool*, so they are not discussed further here.

This completes our discussion of the Text Editing mode of *cmdtool* as well as of *textedit*. We now discuss a few more details about Text Selections, and show you how to start a *cmdtool* window to run one specific program instead of running a shell.

5.8 More About Text Selections

Since the MIT xterm program is so widely used, you may on occasion need to transfer text between cmdtool and xterm, or vice versa. This section discusses the details.

5.8.1 Copying and Pasting between XView and MIT Clients: xcutsel

You can select text in *cmdtool* (or *textedit*, or any OPEN LOOK client that has a text field or a text window), then move the pointer to an MIT *xterm* and click the middle button. Your text will be pasted into the *xterm*. This technique also allows you to copy text into *xedit*, *xclipboard*, and other MIT-style clients.

But to copy from an MIT *xterm* (or *xedit* or *xclipboard*) into an XView OPEN LOOK client, you have to use the following procedure:

- Select the text in the *xterm*, using *xterm* conventions (sweep with Button1 (left), extend if necessary with Button3 (right button));
- Move the pointer into the *xclipboard* main window;
- Paste the text into the *xclipboard* window using the *xterm* conventions (Button2, middle button, pastes text);
- Move the pointer into the OPEN LOOK window;
- Paste the text using the *Paste* button or the *Paste* item from the *Edit* menu.

5.8.2 Saving Multiple Selections: xclipboard

Xclipboard lets you have multiple selections. Its *New* button creates a new selection; once you have created more than one, the *Next* and *Previous* buttons move you among them.

5.8.2.1 Editing Text Saved in xclipboard

It is possible to edit text in an *xclipboard* window, but you should be aware that it uses the *xedit* conventions, which are similar to *textedit*, but not identical. Refer to the discussion of *textedit* in *Chapter 5, The cmdtool/shelltool Terminal Emulator,* and of *xedit* in Chapter 8, *Other Standard Clients*, for more details.

5.9 Running a Program in a Temporary cmdtool Window

Normally you start up a *cmdtool* window and keep it running for a long time, typically the duration of your session. Each such *cmdtool* has its own copy of your login shell (more precisely, the program named in the SHELL environment variable) or the shell named in your *.Xresources* file. You can, however, start up an *ad-hoc* copy of *cmdtool* just for the duration of a single command. All you do is put the name of the command (and any arguments to it) *after* any generic XView arguments:

```
% cmdtool [generic-arguments] command [command-arguments]
```

For example, if you want to run *more* on a file named *temp* in a window that will disappear when you quit (or reach end of file), you can do so as follows:

```
% cmdtool more temp
```

The generic arguments (such as -display) must appear before the command, and arguments to the command must be last. For example, to run the above command with the output appearing on the second screen of a two-headed display,

```
% cmdtool -display unix:0.1 more temp
```

Be aware, though, that if the command terminates without reading from the terminal (as would be the case if file *temp* were shorter than one screen full of text), there is a good chance that it will terminate before the window is fully displayed, in which case you will not see any output. One way around this is to use a *sleep* command, as in

```
cmdtool more temp \; sleep 600
```

which will sleep for up to ten minutes (600 seconds) after the *more*. You can interrupt this by a Control-C (or your interrupt character) in the temporary window. Another way is to read from the window's standard input after the command terminates:

```
cmdtool cat myfile \; echo Press Return \; read junk
```

This will wait until you press return before terminating the window.

5.10 Cmdtool as a Console Window

In the olden days of UNIX, there was a special printing terminal set aside for the console operator, and it had the special name */dev/console.* This file is used by many programs to log messages on, and the file still exists in modern UNIX, even though the hardcopy terminal is gone from workstations. On most modern UNIX systems, */dev/console* is a pseudo-device that causes the log messages (actually anything written to */dev/console*) to appear "somewhere else". That "somewhere" may be on a printer, in a file, or on another terminal. In the case of workstations with graphic screens, it is usually the graphic screen, so that the workstation user will see any "important" messages that are sent to /dev/console. But in the case of a windowing system such as X, it is not a good idea to have programs writing directly to the screen, as this would appear over top of various windows, and jumble up the screen contents (see Figure 5-28).

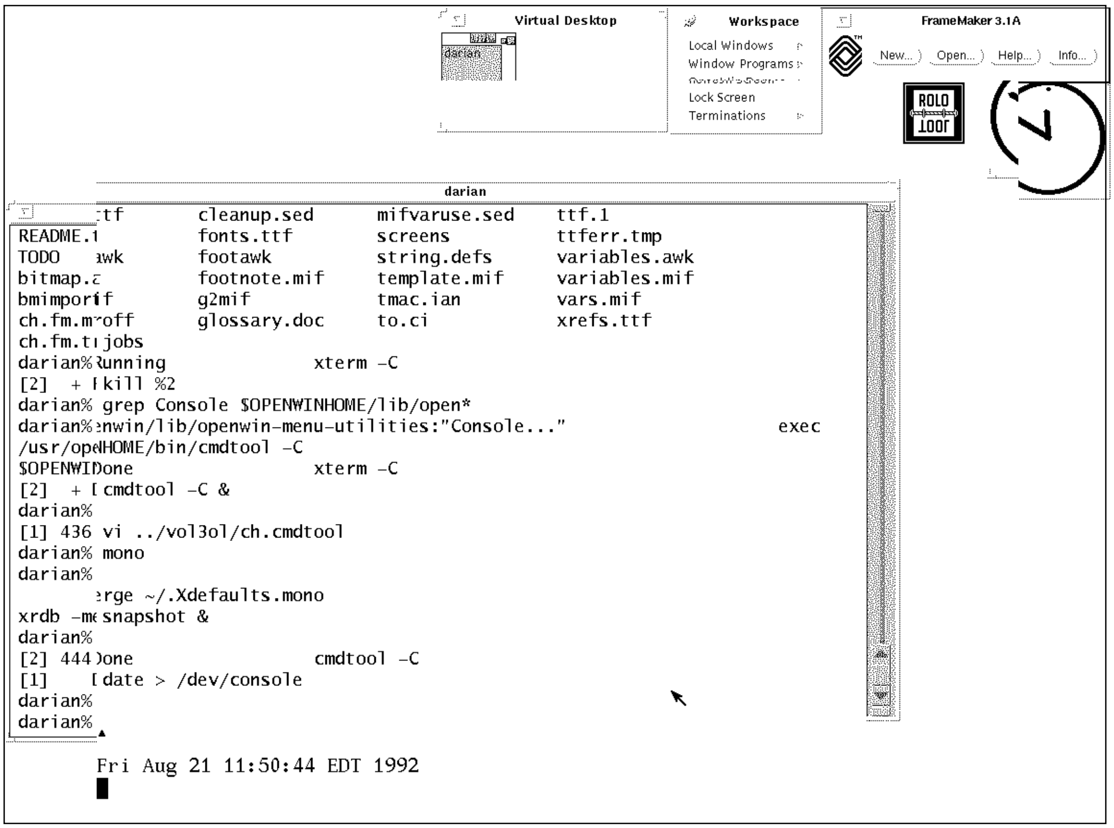

Figure 5-28. A screen with console messages over windows

To solve this problem temporarily, you must *refresh* or *repaint* the screen, that is, repaint the root window and then have each client program repaint its window(s). On OpenWindows this is available in the default menu under *Utilities* on the workspace menu, where it is the default item so you need only click SELEECT on Utilities in the root menu. With other window managers your mileage may vary.

It is better to *prevent* this problem, which you normally do by making a terminal emulator "become the console". The *cmdtool* program (and other terminal emulators such as *xterm*) accept a -C option to make them the console. This is normally specified for the first or last terminal emulator that your session scripts start up, and it is often a small (6 lines by 80 characters) window, to allow you to see messages but not use up much screen space. For example, the default OpenWindows start-up script (*.openwin-init*) has this entry:

```
$OPENWINHOME/bin/cmdtool -Wp 0 0 -Ws 590 77 -C
```

The -Wp and -Ws are standard command line options for position (upper left corner) and size (small) of the window. The -C makes this instance of *cmdtool* become the system console window.

A special-purpose program, *contool*, is available as contributed software. *Contool* is optimized for use as a console; you cannot type commands into it, but it allows you considerable flexibility in filtering out the more common but uninteresting messages that may appear on your console. It can arrange to write some or all messages into a log file, and can open (de-iconify) itself automatically when certain messages appear. See the reference manual in Part Three of this guide for details on this useful (con)tool.

5.11 Cmdtool/Shelltool/Textedit Menus Reference

The chart in Figure 5-29 shows the menus available, starting in *shelltool* mode. For example, turning *Scrolling on* sets the mode to *cmdtool*, where the Term pane menu lists History, Edit, etc. This is the same menu you get if you start off in *cmdtool* mode. Similarly, setting *File Editor Enable* turns on *TextEdit* mode, wherein the menus are the same as if you had started off by running a *textedit* window.

```
ShellTool's Term Pane
Enable Page Mode
Copy
Paste
Enable Scrolling --> CmdTool Mode
        Cmdtool's Term Pane
        History -> Mode -> Editable/Read-Only
            -> Store log as new file...
            -> Clear log
        Edit -> Again
            -> Undo -> Undo Last Edit
                    -> Undo All Edits
            -> Copy
            -> Paste
            -> Cut
        Find-> Find & Replace...
            -> Find Selection-> Forward
                        -> Backwards
            -> Find marked text ...
            -> Replace Field-> Expand/Next/Previous
        Extras-> Format
            -> Capitalize ->
            -> Shift Lines ->
            -> Pretty Print C
            -> Insert Brackets ->
            -> Remove Brackets ->
        File Editor --> Enable
                    Textedit's Text Pane
                    File-> Load File
                        -> Save Current File...
                        -> Store as New File...
                        -> Include File
                        -> Empty document
                    View-> Select Line at #...
                        -> What line #?
                        -> Show caret at top
                        -> Change Line Wrap ->
                    Edit
```

```
Find (same as Find above)
Extras (same as Extras above)
```

Figure 5-29. Shelltool/Cmdtool/Textedit Menus Reference

5.12 Other Terminal Emulator Programs

Now we've discussed the *cmdtool* terminal emulator in detail. There are many other terminal emulators that you should at least be aware of.

5.12.1 Xterm

Xterm, the MIT standard terminal emulator, is widely used in some parts of the X Window System community. It is discussed in Appendix A, *The xterm/olterm Terminal Emulator*. Since *xterm* is so widely used, a brief comparison is in order: please refer to Table 5-2.

Table 5-2. Shelltool/Xterm Comparison

Cmdtool/Shelltool	Xterm
Cmdtool has the advantage that it complies fully with the OL specification.	Widely used.
Cmdtool works with the OPEN LOOK cut and paste, drag and drop, and other mechanisms.	Xterm has a much simpler copy-and-paste mechanism: you select with button1, extend with button3, and paste the selection with button2, instead of with function keys or a menu.
Cmdtool has a better history mechanism, letting you revise the shell history, save it into a file, etc.	*Xterm* has a limited history: you can select from any previous line or lines, but not change them.
Because of its history mechanism, earlier versions of *cmdtool* had problems with command interpreters that play games with *stty* settings, such as *csh* with "set completion" on, *tcsh, bash, ksh* with "set -o vi/emacs", etc. These are fixed now.	*Xterm* is faster. It passes interrupt characters (^C, ^Z) much more quickly, and it uses less CPU time in normal operation.
The version of *cmdtool* in OpenWindows 3.0 has problems when scrolling text while a window partly overlaps it; some of the text does not get redisplayed. This will probably be fixed soon, possibly by the time you read this.	Not a problem.
Geometry size in pixels (700x500)	Geometry size in characters (80x24)

5.12.2 Others

jet is an experimental terminal emulator that is distributed in both source code and binary forma as part of OpenWindows 3.0. A related program is *jed*, the jet-like editor. You could say that *jed* is to *jet* as *textedit* is to *cmdtoo*l; the terminal emulators feature simple editing, and the editors work only on text file but have more functionality.

Some commercial X packages include their own terminal emulators. Examples include DECterm in Digital Equipment's DECWindows and AIXterm in IBM's AIX version of UNIX. These are discussed in the appropriate vendor documentation.

Now let's turn our attention to some other X clients. First we'll discuss some additional features of the OPEN LOOK Window Manager. The next two chapters deal with useful tools; they cover the OpenWindows DeskSet tools and the MIT "standard" clients that should be in any release of The X Window System. This is followed by a chapter on Graphics Clients that mentions MIT clients, OPEN LOOK clients, and some free software clients.