

2

Getting Started Using X

This chapter helps you learn to start the X server, the *fvwm** window manager, and at least one *xterm* (terminal emulator) window. The *fvwm* and *xterm* processes may be started automatically when you log in, or you may have to start them manually. The chapter also describes how to manage windows, start additional clients, and run clients on remote machines. Finally, you'll learn in more detail how to manage your *xterm* windows.

Because both the X Window System itself and the window manager are highly customizable, almost everything about the user interface is subject to modification. What's described here might work somewhat differently on your system. On the other hand, your system might work the same way but you'd like it to behave differently. This chapter describes the default behavior of a standard, uncustomized system, which we'll define as using the *fvwm* window manager and one *xterm* window. In Chapter 12, *Setting Resources*, Chapter 13, *Using Fonts with X*, and Chapter 14, *Specifying Color*, you'll learn how to configure the system to meet your needs and desires.

To start using the X Window System, you must do three things:

- Start the X server
- Start a window manager
- Start at least one *xterm* terminal emulator

Starting X

Depending on how X is run on your system, both the initial screens as well as the way you log in will be slightly different. On many systems, you may log in at the command line

* The *fvwm* window manager has been superseded by *fvwm2*, described in more detail in Chapter 5, *FVWM*. However, we still refer to *fvwm* except when used explicitly in code.

and start X manually. This is particularly common when you are just setting your system up and testing it. Even after that, many people like the flexibility that running X manually provides.

On other systems, the display manager, *xdm*, starts X and keeps it running. If your system is set up to use *xdm*, you log on in a special window provided for that purpose. If *xdm* is running, you should not have to start X manually.

X is easy to customize. There are countless command options and startup files that control the way the screen looks and which menus a program displays. If X has been customized on your system, or you are trying X using someone else's system or login account, things may not work exactly as described here. In particular, if you are using one of the newer window managers, or the KDE or GNOME desktop environments, you will find that some (or even much) of what we describe here works differently on your system. If you want, you can skip directly to Part II. We recommend, however, that you at least look through this chapter and Chapter 3, *Selection of Useful X Clients*, to get an understanding of how X works and what some of its basic capabilities are. From there, you can go on to make the best use of the capabilities of your own environment.

Starting X Manually

To start X manually, either boot your Linux system to the command line and log in, or simply log in if the system is already running. When you see the command-line prompt, enter:

```
| startx
```

startx is a wrapper around another program, *xinit*, which actually starts the X server. If you want to start X at a particular screen resolution, you can specify the resolution as an option. For example:

```
| startx -- -bpp 32
```

The double-dash argument passes subsequent arguments (in this case *-bpp 32*) directly to *xinit*. The argument itself indicates a color resolution of 32 bits per pixel, resulting in TrueColor mode.

When X starts, you'll be logged in to the account from which you ran *startx*.

Logging in With xdm

The display manager, generally *xdm*, is a client that is designed to start the X server automatically and to keep it running. In its most basic implementation, the display manager is a graphical version of the login programs *getty* and *login*, which run on a standard text terminal to display the login prompt, keep the login process running, prompt for the user's login name and password, and manage a standard login session.

If the display manager, *xdm*, is running on your system, you'll see a window with login and password prompts when you turn on your monitor. The example in Figure 2-1 shows a Red Hat Linux *xdm* window.

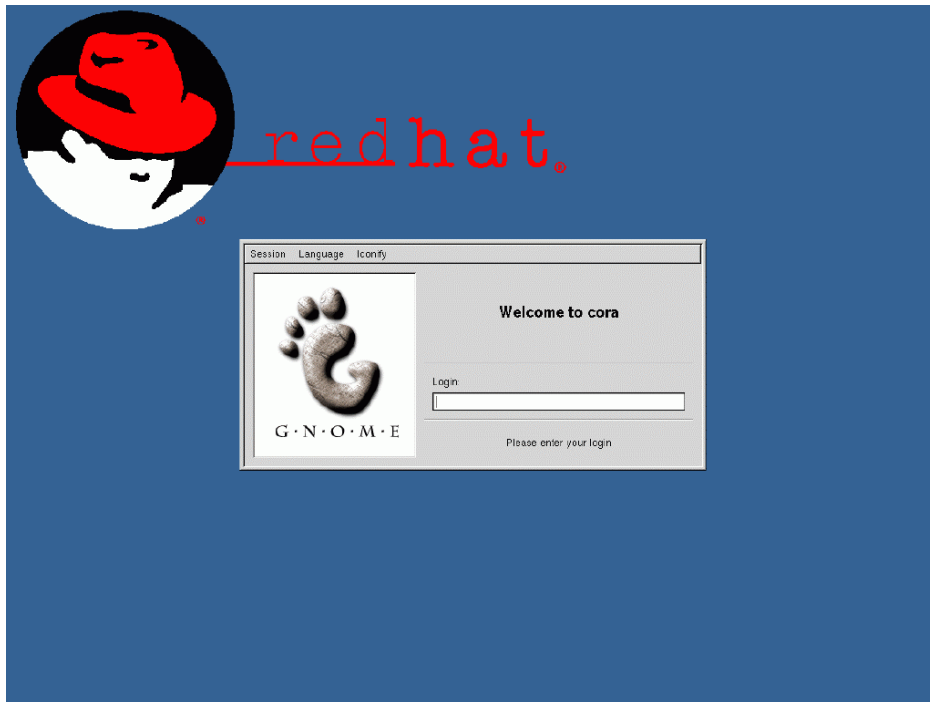


Figure 2-1: xdm login window

Without any customization, the display manager executes a standard login session, starting the window manager and possibly putting up the first *xterm* window. The *xterm* window is displayed in the upper-left corner of the screen. Once the window manager (*fvwm* in this case) is running, you can proceed to type commands in the *xterm* window, open new windows, etc. If the *xterm* window does not start automatically, you can start one by clicking on the root window with the first pointer button and selecting *xterm* from the menu that appears.

Note that in addition to *xdm*, alternative display managers are available. In particular, the desktop environments, described in Chapter 8, *Using GNOME*, and Chapter 9, *Using KDE*, come with replacements for *xdm--gdm* and *kdm* respectively--that you might prefer to use in those environments. These replacements behave like *xdm* but have the look and feel of their desktop environment. See Chapter 11, *X Display Managers*, for more detailed information.

Exiting Your X Session

At some point, you'll want to end your X session. You do this by terminating the window manager. With *fvwm*, click on the root window with the first pointer button to get the root menu, and then select the option Exit Fvwm. That displays a submenu, which offers several options; selecting Yes, Really Quit terminates *fvwm*. (Other options let you restart *fvwm* or start another window manager.)

If you started your X session with the *startx* command, leaving *fvwm* stops the X server and puts you at the command line in your login session. If you used *xdm* to log into X directly, stopping *fvwm* resets the X server and redisplayes the *xdm* login window.

Some Useful X Keyboard Shortcuts

The X Window System provides some keyboard shortcuts that you should know about:

Ctrl-Alt-Backspace If you ever run into a situation where the normal way of leaving X by stopping the window manager doesn't work, you can use the Ctrl-Alt-Backspace combination to kill the X server. Note that you'll lose any unsaved changes in applications running at the time,

Ctrl-Alt-Delete If you get into more serious trouble and need to reboot your system, Ctrl-Alt-Delete will do that. It's a brute-force technique and should only be used when all else fails, but it does the job.

Ctrl-Alt-KP+ and Ctrl-Alt-KP- The combination of the Ctrl, Alt, and the keypad plus and minus keys let you change your screen mode while you are in X. Ctrl-Alt-KP+ cycles forward through modes defined in the Screen section of XF86Config while Ctrl-Alt-KP- cycles backward. Since it only works on modes that are defined for your screen, these key combinations have no effect if you only have one mode defined.

Ctrl-Alt-Fn Linux provides the ability to switch between *virtual consoles*. From the Linux command line, you can use the Alt-Fn key combination, where Fn is one of the function keys on your keyboard, to move between virtual consoles and establish multiple login sessions. When you are in X, you can still move between those virtual consoles, except you use the key combination Ctrl-Alt-Fn to get to the console you want. You can then use Alt-Fn to move between virtual consoles directly, or Alt-F7 to return to your X session. The number of consoles varies depending on your distribution, but generally at least four (Alt-F1 through Alt-F4) are available.

Starting the Window Manager

It is unlikely that you will ever have to start your window manager manually. Most, if not all, Linux distributions come with a default window manager configured to start when you log into X. A few years ago, the default was likely to be either *twm* or *fvwm*. Recently, however, the major distributions are more likely to come configured to run KDE or GNOME. KDE by default uses its own window manager, *kwin*, while GNOME uses Enlightenment or Sawfish as its default, depending on the distribution. However, in this chapter, we'll talk about *fvwm*, as it has the basic features we want to discuss.

Once the window manager and an *xterm* window are running, you can do some work. Typically you will run more than one application and so will be working with several windows. The following sections describe basic features of windows and explain how to move, size, and arrange them.

Input Focus: Typing in an *xterm* Window

Now that you've started the X server, the first *xterm* window, and the *fwm* window manager, you'll want to start entering commands in the *xterm* window. However, if you simply start typing, you may find that the keystrokes do not appear in the window--or anywhere else on the display.

You can provide input to only one window at a time. X does not automatically know which window you want to type in, even if only one window appears on the display; first you have to select the window by setting the input focus on that window. The window to which input is directed is often referred to as the *active* or *focus* window.

The default method for moving the focus to a window when *fwm* is running is to simply move the pointer so that the cursor symbol is within the desired window. This focus policy is commonly referred to as *auto-focus*, *pointer* focus, or *real-estate-driven* focus. The alternative policy is to use *click-to-focus* or *manual* focus. This requires you to move the cursor to the window you want to select and then click the first pointer button.¹ It may be a nuisance to have to remember to click when you change windows, but if you tend to move your pointer accidentally (while you are typing, for example, or looking through papers), click-to-focus can save you from typing in the wrong window.

Until you select an *xterm* window as the window to receive input, any keystrokes are lost. You can tell which window has the focus by changes in the appearance of the display. First, the titlebar is highlighted or changes color in some way, depending on the version of *fwm*, the color resources available on your system, and how they are configured. In any case, the active window's titlebar looks different from the titlebars of other windows on the display, and the window border is highlighted.

The cursor symbol also changes. When the pointer rests on the root window, the cursor generally appears as a large X. When you move the pointer into an *xterm* window, the cursor changes to what is known as the I-beam cursor symbol, which looks like a very thin letter "I".

In the *xterm* application window itself, there is also a rectangular text cursor, which changes from an outline to a solid box. Finally, depending on how *fwm* is configured, the window may be raised if it was hidden under another window.

Once you've selected the *xterm* window as the active window, the characters you type appear on the current command line, regardless of where in the window the I-beam cursor is.

You can configure *fwm* to raise the focus window automatically. This behavior is controlled by the *fwm* variable `AutoRaise`, which you can add to the *fwm* configuration file (`.fwmrc`) in your home directory. See Chapter 5, *FVWM*, for more information.

¹ Some window managers also support a focus mode known as *sloppy-focus* or *semi-auto-focus*. This focus mode lets you move the pointer from the active window to the root window without losing focus, but if you move it to another non-root window, the focus also moves to that window. *fwm* does not support sloppy-focus.

Using the Pointer

The cursor on the screen follows the pointer's movement on the desktop. You use the pointer to select a graphical element on the screen, such as a window, icon, or command button. X generally assumes that your pointer is a three-button mouse. If your mouse only has two buttons, you can configure three-button emulation by setting `Emulate3Buttons` in the Pointer section of the file `XF86Config`. (See Chapter 10, *XFree86*, for information on configuring XFree86). Turning on three-button emulation tells X that simultaneous pressing of both mouse buttons represents pointer button 2. (By default, the left button is button 1 and the right button is button 3.) Because it's difficult to press two buttons at exactly the same moment, you can also set the `Emulate3Timeout` option to give yourself time to press the buttons. The default timeout is 50 milliseconds; with this default, as long as you press the second button within 50 milliseconds of the first button, the system treats it as a simultaneous press. Here's an example of the `XF86Config` settings:

```
Emulate3Buttons
Emulate3Timeout 50
```

If 50 milliseconds isn't enough time, you can increase the timeout to a larger number.

If you want to reconfigure the buttons (for example, you want to swap buttons 1 and 3), use a client called `xmodmap`. An example of the use of `xmodmap` is shown in Chapter 13.

The following sections describe how to perform the most basic window management functions.

Managing Windows Using the Titlebar

Figure 2-2 shows an `xterm` window with the titlebar provided by `fvwm`. The titlebar itself and its component parts are tools that allow you to manage the window using the pointer.

[Illustration dept.: Please cut off the bottom two-thirds of the following figure with a squiggly line. We only want to show the titlebar and the command lines.](#)

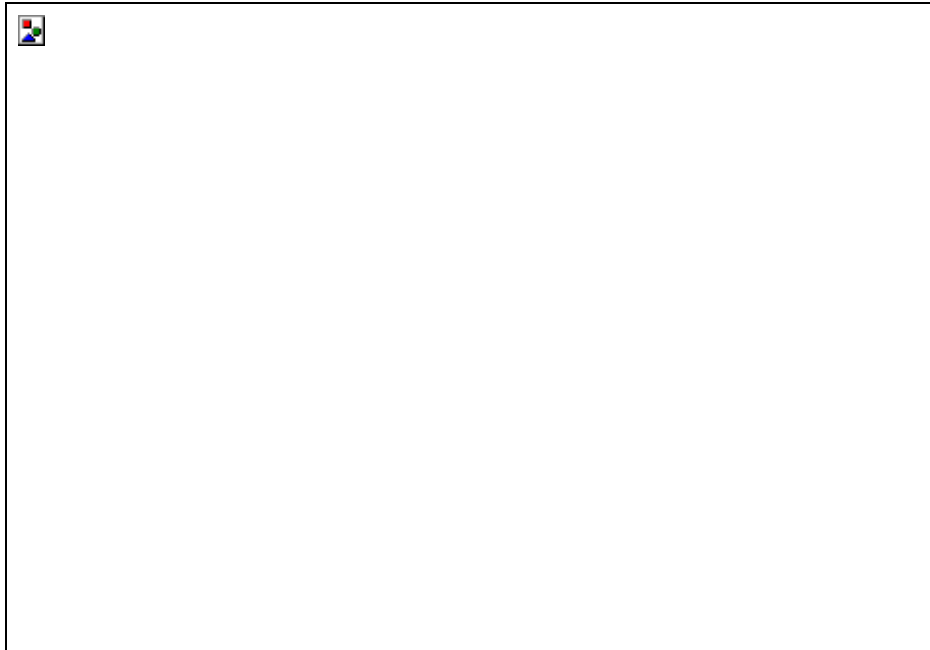


Figure 2-2: An xterm window with fvwm titlebar

The titlebar spans the top of each window on the display. It contains the name of the application (e.g., *xterm*), two buttons on the left for a drop-down menu and to “stick” the window so it shows up on any virtual screen, and two buttons on the right to iconify and maximize the window. Like just about everything else in X, the exact configuration of the titlebar is configurable, and you may find that the default configuration of your titlebar looks different. In particular, some distributions of *fvwm* don’t include the sticky button.

Whenever you move the pointer onto one of the buttons, the pointer changes to a small hand cursor, indicating that you can push the button. If you move the pointer onto any other part of the titlebar, it changes to an arrow cursor.

Though it’s not apparent from Figure 2-2, you can perform most window management functions by using the pointer on various parts of the titlebar. The following sections explain how to iconify, move, raise, lower, and maximize windows using the titlebar.

Converting a window to an icon

An icon is a symbol that represents a window in an inactive state. As we’ll see later, the virtual desktops available with *fvwm* and other window managers provide another way to solve this problem, but you don’t always want to have to switch desktops. Having some windows iconified also frees you from constantly shuffling the stacking order by raising and lowering windows.

Click on the *iconify* button on the titlebar (the triangle pointing down, second in from the right) to convert a window to an icon. If you don’t immediately see the icon you just made, look under some of your other windows—for some reason, by default *fvwm* places icons under active windows. You can solve this problem by defining an *IconBox*: a region of the screen where icons are placed, at a location of your choice (see Chapter 5 for details).

Converting an icon to a window

To convert an icon back to a window (deiconify it), place the pointer on the icon and click the first (usually left) pointer button. The window is redisplayed in its previous position, with the focus in that window so you can immediately use it.

Raising a window or icon

If you have several windows open on the display, some of them probably overlap. In order to work with a window that is all or partially covered by another window, you can raise it by placing the pointer on any part of the titlebar except the iconify and resize buttons or anywhere on the border, and clicking the first pointer button. Icons may be obscured by windows or other icons on the display. By default, the only way to raise an icon using the pointer is to deiconify it--which also raises it to the top of the stack. The bigger problem may be finding the icon when it has been buried under a window. One solution is to lower the obscuring window instead.

Lowering a window

Lowering a window in *fwm* (sending it to the bottom of the window stack) is the same as raising it, except that you click the third pointer button instead of the first. While it's usually easier to raise the window you want, you might want to lower a window that is totally obscuring the window or icon you want.

Moving a window

To move a window, click the first pointer button in the middle section of the titlebar (which contains the application name and the highlight region) and drag the window to the new location. This location can be on the same virtual desktop, or with some window managers such as *fwm*, you can drag the window to a new virtual desktop by continuing to drag it off the edge of the screen in the direction of the new desktop.

When you move a window, a small box* appears in the upper-left corner of the screen that displays the x and y offset of the upper-left corner of the window in pixels. You can see the offset change as you resize the window; you can use this information to make sure the window ends up exactly where you want it. The offset information is also useful for setting the *-geometry* option if you later want to invoke the program and have the window appear at that same location.

When you use the pointer to move a window, the pointer rests in the titlebar when you are done; if you don't move the pointer subsequently, input remains focused on the window you've moved. Also note that moving a window raises it to the top of the stack.

Moving an icon

Moving an icon is similar to moving a window. To move an icon, place the pointer on the icon, press the first pointer button, move the icon by dragging the pointer, and release the button.

Resizing a window

You can resize a window horizontally, vertically, or simultaneously in both directions. If you click the first pointer button on a corner of the window, you can

* The small box is a feature provided by the window manager. While most modern window managers, including *fwm*, provide this feature, not all older window managers (e.g., *twm*) do.

resize the window by pushing or pulling in the direction you want to go. Move the pointer diagonally to modify both height and width.

Depending on your distribution, you may be able to resize the window either horizontally or vertically by clicking the first pointer button on the border of the side you want to move, and then pushing or pulling the side of the window out to the size you want. The other sides of the window remain stationary.

As with moving a window, you see a small box in the upper-left corner of the screen when you click the pointer button. This time, though, the box contains the dimensions of the window, rather than the offset. For most clients, the dimensions are measured in pixels; for *xterm*, they are measured in characters wide by lines high.

Resizing a window also raises the window to the top of the stack.

Note that resizing an *xterm* window does not change the dimensions of the text currently in the window. If you make the window smaller, for instance, some of the text may be obscured. Some commands, such as the *vi* editor, adjust automatically to the new screen size. With others, you'll need to reissue the command if you want the output to adjust to the new size.

Maximizing a window

The *maximize* command button on the titlebar (the triangle pointing up, top-right corner) allows you to maximize the window's size. The direction of maximization depends on which pointer button you click:

- Click the first pointer button to maximize the window's vertical size to fit the whole height of the screen.
- Click the second pointer button to maximize the window both vertically and horizontally, thus having the window fill the whole screen.
- Click the third pointer button to maximize the window horizontally.

Re-clicking the maximize button with any pointer button returns the window to its original size.

The window operations menu

Pressing the left pointer button on the dash in the top-left corner of the titlebar activates the window operations menu, which provides a menu-based way to perform the operations we've been describing. The operations available on this menu are described in Table 2-1. The options on this menu vary on different window managers; the ones shown here apply to *fwm*.

Table 2-1 Window operations menu

Function	Description
Move	Allows movement of the window until the left pointer button is clicked to deposit the window at its destination.
Resize	Resizes the window by moving the window's top-left corner.
Iconify	Iconifies the window.
Maximize/Normal	Maximizes the window to the screen's full size. If the window is already maximized then the window returns to its original size.
(Un)Stick	A window is made "sticky" which means it's visible in all the virtual

	screens. If a window is already “sticky” it is made “unsticky.” This results in it only being visible in the current virtual window.
Window list	Shows a list of all the virtual screen’s windows and their sizes in either pixels or the window’s dimension sizes.
Close	Closes the window gracefully.
Kill	Kills the window.

Creating New Windows

Once you focus input on the first *xterm* window, you can enter commands. For example, you can enter a command to open a second *xterm* window by typing the following command at the prompt in the first *xterm* window:

```
| xterm &
```

(The ampersand, *&*, causes the command to run as a background process so you can continue to use that window. See the section “Starting additional clients” for a more complete explanation.)

If there’s room for the new window without it having to overlap an existing window, *fvwm* places it there for you. Otherwise the pointer symbol becomes an upper-left corner cursor and the outline of a window appears, allowing you to position the new window wherever you want it. You can move the corner cursor to the desired position on your screen and *click* the first pointer button. A new *xterm* window then appears at that location on your screen, displaying the prompt from the shell you are presently using.

When you use the pointer to place a new window, the pointer rests in the upper-left corner of the window when it is displayed; thus, input is always focused on that window.

Starting Additional Clients

Now that you know the basics of managing windows, you can start other X clients just as you can start another instance of *xterm*. The following sections describe how to open more client windows, place them on the display in convenient positions, and take advantage of X’s networking capabilities by running clients on other machines. (Each window manager also has features for starting clients without having to enter a command on the command line. We’ll learn about those features in the chapters about the window managers and desktop environments.)

First, let’s digress a minute and talk about background processes. In a standard terminal session (i.e., on a hardware terminal or in an *xterm* window), when you run a command, you lose control of the terminal or window until the command has finished processing. Running a process in the *background* means that you regain control of your terminal and can continue working while the process is running. To run a process in the background, you enter an ampersand (*&*) after the command:

```
| ghostview myfile &
```

When you are running X, starting a new client means that you are going to generate a new window on the screen. If you do that without specifying the *&*, you will lose the use of the

xterm window for the whole time that new window is in existence. (There are some exceptions to this, for clients that simply run, do their job, and exit--e.g., *xdpyinfo*.)

Bearing that in mind, when you want to start a client at the command-line prompt in any *xterm* window, type the name of the client followed by an `&` to make the client run in the background. For example:

```
| xclock &
```

This command starts a clock application, as shown in Figure 2-3. Since you want a clock to remain active for the duration of the session, you run it in the background so you can keep using the *xterm*.



Figure 2-3: The *xclock* window

Though it's easy to move windows on the display, manually positioning every window is not particularly convenient. The next section describes two command-line options that let you specify window location and size.

Command-line Options

The size and position of a window are referred to as its *geometry*; you can set these attributes using the *-geometry* option. In addition, you can use the *-display* option to specify the screen on which a window should be created. (A common use for *-display* is to run a client on a remote system and display the window on your screen.) The next sections describe these options in detail.

Window geometry: specifying size and location

The command-line option to specify a window's size and location has the form:

```
| -geometry geometry
```

The *-geometry* option can be abbreviated *-g*, unless the client accepts another option that begins with *g*.

The value of *geometry*, referred to as the *standard geometry string*, has four numerical components, specifying the window's dimensions and location. The standard geometry string has the syntax:

```
| widthxheight+xoff+yoff
```

The first half of the string specify the *width* and *height* of the window. Many application windows are measured in pixels, but other units are used if they are more meaningful in terms of the application. For example, as we saw earlier, an *xterm*'s dimensions are measured in characters and lines: an *xterm* window is some number of characters wide by some number of lines high (80 characters wide by 24 lines high by default).

The second half of the geometry string gives the location of the window relative to the horizontal and vertical edges of the screen. Imagining the screen to be a grid (where the upper-left corner is 0,0), the x offset, *xoff*, and the y offset, *yoff*, represent the x and y coordinates at which the window should be displayed, measured in pixels.

The term pixel comes from *picture element*; it is the smallest element of a display surface that can be addressed by a program. Think of a pixel as one of the tiny dots that make up a graphic image, such as that displayed by a terminal or a television. The number of dots (or pixels) per inch of screen determines the screen's *resolution*. The more dots per inch, the higher the resolution (and, hypothetically, the sharper the picture).²

Since a pixel is a tiny unit of measurement, gauging sizes and distances in pixels takes some practice. The *xdpyinfo* client, described further in Chapter 3, can be used to find such information. *xdpyinfo* also tells you the screen's resolution in dots per inch, and a lot more. Such information may be useful, because the size and location of client windows is related to the size and resolution of your screen. For example, if you specify a window with dimensions of 125 x 125 pixels, the apparent size of that window depends on the screen and its resolution. Imagine how large it looks on a screen sized 640x480 and how small it might look on a 1280x1024 screen.

When you actually use the geometry option to specify a window's size and location, you can specify any or all elements of the geometry string. Incomplete geometry specifications are compared to the application's default settings and missing elements are supplied by these default values. For example, every client application has a default size. If you run an *xterm* window with the geometry option and specify a location but no dimensions, the application's default size of 80 characters by 24 lines is used.

For now, let's just specify a window's location and accept the default size. The x and y offsets can be either positive or negative. If you specify positive offsets, you're positioning the left side and the top of the window. Negative offsets position the right side and the bottom of the window. The possible values for the x and y offsets and their effects are shown in Table 2-2.

Table 2-2: Geometry Specification of x and y Offsets

Offset	Description
<i>+xoff</i>	A positive x offset specifies the distance by which the left edge of the window is offset from the left side of the display.

² There are other factors that determine the "picture quality" of a display, including "depth," or the number of bits per pixel. Depth relates to how many colors that can be simultaneously displayed. For more information, see the section "How Many Colors Are Available" in Chapter 14.

<i>+yoff</i>	A positive y offset specifies the distance by which the top edge of the window is offset from the top of the display.
<i>-xoff</i>	A negative x offset specifies the distance by which the right edge of the window is offset from the right side of the display.
<i>-yoff</i>	A negative y offset specifies the distance by which the bottom edge of the window is offset from the bottom of the display.

For example, the command line:

```
| xclock -geometry -10+10 &
```

places a clock of the default size in the upper-right corner of the display, 10 pixels from the right and top edges of the screen.

To place a window in one of the four corners of the screen, flush against its boundaries, use the x and y offsets shown in Table 2-3.

Table 2-3: Geometry Specification for Screen Corners

Offset	Window Location
+0+0	Upper-left corner
+0-0	Lower-left corner
-0+0	Upper-right corner
-0-0	Lower-right corner

If you want a window placed away from one or both edges of the screen, the guesswork starts. You can experiment by specifying some offsets and seeing where the client windows end up on the screen.

You can also place some windows in different positions by dragging and then determine their geometry specifications. One way to find the geometry is by placing the cursor in the titlebar of the window as though you were going to move it; the coordinates are then displayed in the small box, as we saw earlier.

Another technique is to use the *xwininfo* client, which displays information about a window. When you enter the *xwininfo* in an *xterm*, you see a message that says:

```
| xwininfo: Please select the window about which you would  
| like information by clicking the mouse in that window.
```

and the cursor turns into a large plus sign (+). Move the cursor onto the window whose geometry you want to check, and you'll see something like this:

```
| xwininfo: Window id: 0x140000e "ruby"  
| Absolute upper-left X: 438  
| Absolute upper-left Y: 60  
| Relative upper-left X: 0  
| Relative upper-left Y: 0  
| Width: 499  
| Height: 628  
| Depth: 24  
| Visual Class: TrueColor
```

```
Border width: 0
Class: InputOutput
Colormap: 0x23 (installed)
Bit Gravity State: NorthWestGravity
Window Gravity State: NorthWestGravity
Backing Store State: NotUseful
Save Under State: no
Map State: IsViewable
Override Redirect State: no
Corners: +438+60 -87+60 -87-80 +438-80
-geometry 80x48-82+37
```

This gives you possibly more information than you ever wanted to see about the window, including (in the last two lines) the information you want: the offsets of the corners and the geometry of the window. You can also specify *xwininfo* options to customize the output (try *xwininfo -help* to see a list of options).

To specify the *size* of a window, you need to know whether the client uses pixels, or characters and lines. We've already seen that *xterm* uses characters and lines. Thus, to get a large terminal window, say 100 characters wide by 55 lines high, you could use this geometry specification:

```
| xterm -geometry 100x55-0-0 &
```

This command creates a large *xterm* window in the lower-right corner of the screen.

Most other the standard clients are measured in pixels. For example, the default *xclock* size is 164 pixels square (exclusive of the window manager titlebar). A client's default dimensions may be given in its manpage, but you'll probably need to experiment with specifying sizes (as well as locations) on your display.

The geometry option is not necessarily the only way to specify window size and location. Most X clients, including *xterm*, allow you to set the size and location of a window (and often its icon or an alternate window) using resource variables (in an *.Xresources* or other resource file). We'll introduce some of the basics of specifying resources later in this chapter and in more detail in Chapter 12.

With window managers that include virtual desktops, you can specify a geometry that is on a different desktop, off the visible portion of your display screen. Imagine entering the following command on your 1024x768 screen.

```
| xclock -geometry 500x500+2000+400 &
```

You're sure the process is running but it's nowhere to be seen. *fvwm*'s virtual screen manager will give you a hint.

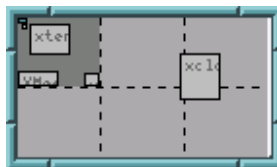


Figure 2-4: *fvwm*'s virtual screen manager

The *xclock* is so far to the right that it's on a different virtual screen—in fact, it overlaps several screens.

Specifying the display

We have not yet taken advantage of X's networking capabilities. Remember that X allows you to run a client on a remote machine. Generally, the results of a client program are displayed on a screen connected to the system where the client is running. However, if you are running a client on a remote system, you probably want to see the results on your own display (connected to a local server).

Once a remote client is running, using the client is no different than using a local client. You can display the application window on your own screen, enter input using your own keyboard and pointer, and read the client's output in the window on your screen--all while the actual client process is running on the other machine.

In the simple case, when you are running a client locally and also displaying the output locally, the client accesses the DISPLAY environment variable to determine which physical display to connect to (for most clients, "connecting" is equivalent to opening a window). The DISPLAY variable is set automatically by *xinit* or *xdm* when you log in, and *xterm* inherits it.

However, a client running on a remote machine does not have access to the local DISPLAY variable, which is stored in the local shell. In that case, you must use the *-display* option to tell the client process where to display its window. Think of *-display* as a pointer to the physical display on which you want the window to appear. Like *-geometry*, the *-display* option is recognized by most X clients; it tells the client on which server to display results (i.e., create its window). The *-display* syntax is:

```
| -display [host]:server[.screen]
```

For example:

```
| -display kansas.ora.com:0.0
```

-display can be abbreviated as *-d*, unless the client accepts another option that begins with *d*.

The argument to *-display* is a three-component *display-name*, where the components are:

<i>host</i>	The machine on which to create a window
<i>server</i>	The server number
<i>screen</i>	The screen number

In this context, "host" refers to the Internet address name of the display hardware. For our purposes, it's the name of your Linux system on the network, like *kansas.ora.com*.

"Server" refers to an instance of the X server program, which controls the physical display. An X display may be composed of multiple screens, but the screens share one keyboard and pointer. Most workstations have only one keyboard and pointer and thus are classified as having only one display. Multiuser systems may have multiple independent displays, each running a server program. If there is a single display, as in the case of most workstations and X terminals, it is numbered 0; if a machine has several displays, each is assigned a number (beginning with 0) when the X server for that display is started.

Similarly, if a single display is composed of multiple screens, each screen is assigned a number (beginning with 0) when the server for that display is started. Multi-screen

displays may be composed of two or more physical monitors; alternatively, two screens might be defined as different ways of using the same physical monitor. Each screen is the size of the monitor, and you can only view one screen at a time. In practice, the two screens behave as though they were side by side: you can “scroll” between them by moving the pointer off either horizontal edge of the screen.

Note that the *server* parameter of the display option always begins with a colon, and that the *screen* parameter always begins with a period. If the host is omitted or is specified as *unix*, the local machine is assumed. If the screen is omitted, screen 0 is assumed. Note that this is correct for Linux too: *unix:0.0* works, *linux:0.0* doesn't work.

Although much of the X Window System documentation suggests that any of the *-display* parameters can be omitted and will default to the local node, server, and screen 0 respectively, in the real world, only the *host* and *screen* parameters (and the period preceding *screen*) can be omitted. The colon and *server* are always necessary.

Suppose you're using a single-display workstation and the display has only one screen. The hostname of the workstation is *kansas*. In order to tell a client to connect to a display, you must identify it by its unique name on the network. (You cannot identify your display by the shorthand setting given to it by the X startup program--*unix:0.0*, *:0.0* or some variation.) Let's assume that the complete display name for the workstation *kansas* is:

```
| kansas:0.0
```

That means that the following commands all result in having an *xterm* window started on your display:

```
| xterm &  
| xterm -display unix:0.0 &  
| xterm -display :0.0 &  
| xterm -display kansas:0.0 &
```

Now let's say you want to run an *xterm* window on a faster system called *oz* on your network. In order to run an *xterm* on *oz* but display the window on your local workstation *kansas*, you can run the *xterm* command using a remote shell* (*rsh*) or better, a secure remote shell* (*ssh*):

```
| ssh oz xterm -display kansas:0.0 &
```

The *xterm* process runs on *oz* via the *ssh* command, but you've directed the client to use the display and screen numbered 0 on *kansas*, your local system. Notice that *kansas:0.0* is the complete display name. It's common to specify the complete name in any case, but if *kansas* has only one screen, or if it has multiple screens but you want to use screen 0, you can omit the screen number and the preceding period and specify it as *kansas:0*.

* The command to run the remote process might be different depending on the available networking software.

* See O'Reilly & Associates, Inc.'s *SSH, The Secure Shell: The Definitive Guide*, by Daniel J. Barrett and Richard Silverman, for more information on *ssh*.

Keep in mind that for this process to succeed, the remote client running on *oz* must have permission to “open” the local display on *kansas*. If *oz* has not been granted access to the server running on *kansas*, the window will not open, and you may get an error message similar to this:

```
| Error: Can't Open display
```

If the command fails, try entering the command:

```
| xhost remotesystem
```

in an *xterm* window running locally ; in this example, *remotesystem* is *oz*. Then run the remote shell (*ssh*) again. If that works, you can put the *xhost* command in your *.xinitrc* or *.xsession* file so it runs each time you log in.

In addition to specifying a local display, you can also use the *-display* option to open a window on someone else’s display (if the permissions allow it). You might, for example, want to display a window on another user’s screen for instructional purposes. If you’re working on *kansas*, the following command opens an *xterm* window on the first display connected to *oz*:

```
| xterm -display oz:0.0 &
```

Note that you can only open a window on another display if the server running that display permits the client program to access the display. If *oz* does not allow *kansas* access, the command fails with an error message indicating that the display cannot be opened.

A less than obvious side-effect of using *-display* to run a remote *xterm* is that the option sets the DISPLAY variable for the new *xterm* window--and that DISPLAY setting is passed on to all the client’s child processes. Therefore, once you run an *xterm* on a remote system and correctly specify your own display, you can run any number of clients from that *xterm* and they are all displayed on your screen automatically (no *-display* option is necessary).

In the preceding section, we ran an *xterm* on the remote system *oz*, specifying the local display *kansas:0.0* with the *-display* option. To query the contents of the DISPLAY variable in the resulting *xterm* you can use the command:

```
| echo $DISPLAY
```

The system should echo:

```
| kansas:0.0
```

verifying that the display name has been passed to the DISPLAY variable in the new *xterm* window. You can now run any client on *oz* by entering the command in this *xterm* window and the window is automatically displayed on *kansas:0.0*.

Logging In to a remote system

If you log in to a remote system in an *xterm* window, it’s a good idea to set the DISPLAY variable in the new shell to reflect your local display (assuming there are no security constraints). Then if you run a client process from this window, the new window is placed on your local display and the DISPLAY setting is passed on to all child processes.

When you set the DISPLAY variable from the command line, the syntax varies depending on the shell running. The following command sets the variable using *bash*:

```
| export DISPLAY=kansas:0.0
```

To set the DISPLAY variable with *tcsh*, use:

```
| setenv DISPLAY kansas:0.0
```

Customizing a Program

Command-line options allow you to customize one invocation of a client program. We've already seen how to use the *-geometry* and *-display* options, which are accepted by most clients. Some standard command-line options for setting window features are listed in the following table.

Table 2-4: Some common command-line options

Command-line option	Description
<i>-display</i>	Display on which the client should run.
<i>-geometry</i>	Geometry string for window size and placement.
<i>-fn</i> or <i>-font</i>	Font in which text is displayed.
<i>-bg</i> or <i>-background</i>	Background color.
<i>-fg</i> or <i>-foreground</i>	Foreground color (such as the color of text).
<i>-iconic</i>	Program should be initially iconified
<i>-name</i>	Name under which resource should be found
<i>-title</i>	Text displayed in the title area
<i>-xrm</i>	Resource name and value

[Production – watch out for Word’s funky hyphens in the table above.](#)

Many clients also accept application-specific options; the manpage or other documentation for any program describes the available options. Using a combination of standard and application-specific options, you can tailor a client to look and behave in ways that suit your needs.

Let’s look, for example, at the *xclock* program. Like most clients, *xclock* accepts a variety of options. Some *xclock* options are intended to enhance the clock display aesthetically and some to affect its operation. Taking a look at some of the options should give you a better idea of the flexibility of X.

The following command line runs a custom *xclock* display:

```
| xclock -hd green -hl royalblue -bg lightblue -fg \  
    royalblue -update 1 -chime &
```

X is highly flexible in the use of color, as this example shows. The *-hd* option sets the color of the clock’s hands to green. The *-hl* option provides even more detail, specifying the color of the outline of the hands as royal blue. *-bg* and *-fg* are two of the options accepted by almost all clients; they set the window’s background and foreground color, in this case to light blue and royal blue, respectively.

The `-update` option takes as its argument the frequency in seconds with which the time on the clock should be updated. We've specified that the time is to be updated at one-second intervals. This adds a second hand (also green) to the `xclock` display.

The `-chime` option specifies that the keyboard bell will ring once on the half hour and twice on the hour.

By default, `xclock` displays a traditional clock face (an analog clock). You can also create a 24-hour digital `xclock` with the `-digital` option:

```
| xclock -digital &
```

The result is shown in Figure 2-5:

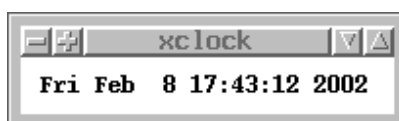


Figure 2-5: Digital xclock display

Options specified on the command line override the default characteristics of a client for that one client process. They also override any personal defaults set in your `.Xresources` file.

Managing xterms

This section describes more about how to use `xterm`, the terminal emulator. You use this client to create multiple terminal windows, each of which can run virtually any program available on Linux.

As we've seen, `xterm` provides you with a terminal within a window. Anything you can do using a standard terminal, you can do in an `xterm` window, including running other X clients.

To take full advantage of X's windowing and networking capabilities, you'll probably want to run more than one application at a time, including multiple `xterms`, perhaps on different systems in your network. Running multiple `xterms` allows you to perform a variety of tasks simultaneously, and to coordinate those tasks. For instance, you can display the contents of a directory in one window while you edit a file in another window. Or you can have a program compiling in one window while you read mail in a second window, and you can use a third `xterm` window to perform tasks based on the contents of the mail.

When you're in an `xterm` window and you start another `xterm` process from the command line, the second `xterm` inherits the environment variables of the first (including the `DISPLAY` setting); the second shell also starts in the same working directory as the first shell.

Each `xterm` has four menus that can be used to select many terminal settings, to change the font size, and to run other commands that affect the `xterm` process. We'll take a look at some of the more useful items on each menu as well as some alternatives to menu items later in this chapter. We'll also consider how to run a program in a temporary `xterm` window that goes away when the program finishes.

But first, we'll see how to remove an *xterm* window; then we'll consider the question of what terminal type to specify for your *xterm*, and finally we'll look at the *xterm* features you'll probably use most frequently: the scrollbar and the text selection mechanism, which lets you copy text from one window and paste it into another.

Exiting an *xterm* Window

When you are finished using an *xterm* window, you can remove it by typing whatever command you usually use to log off your system. Typically, this might be *exit* or CTRL-D. You can also remove it by double-clicking the leftmost titlebar button, or by clicking once on that button and selecting Close from the windows operations menu.

Terminal Emulation and the *xterm* Terminal Type

Anyone who has used a variety of terminals knows that they don't all work the same way. Each time you start an *xterm*, it looks for a terminal type, which tells the system how the window should operate (i.e., the terminal type determines what sort of terminal *xterm* emulates). When *xterm* is assigned an inappropriate terminal type, the window does not always display properly, particularly when using a text editor such as *vi*. An *xterm* window most successfully emulates a terminal when it has been assigned the terminal type *xterm* (i.e., the TERM environment variable is set to *xterm*).

Under normal circumstances, once your Linux system has been installed and X is running and has been configured, you won't have to worry about setting TERM. However, if you find that it isn't getting set properly, you can specify the terminal type in your shell startup script (e.g., *.bashrc*, *.profile*, etc.). *xterm* can emulate a variety of terminal types, including *xterm*, *vt102*, *vt100*, and *ansi*.

Using the Scrollbar

When using *xterm*, you are not limited to viewing the 24 lines displayed in a standard-sized window. By default, *xterm* actually remembers the last 64 lines that have appeared in the window. With the scrollbar, you can scroll up and down through the saved text.

xterm does not automatically run with a scrollbar. To specify the scrollbar use the *-sb* command-line option. Saying:

```
| xterm -sb & |
```

adds a scrollbar sidebar to the *xterm* window, as shown in Figure 2-6.

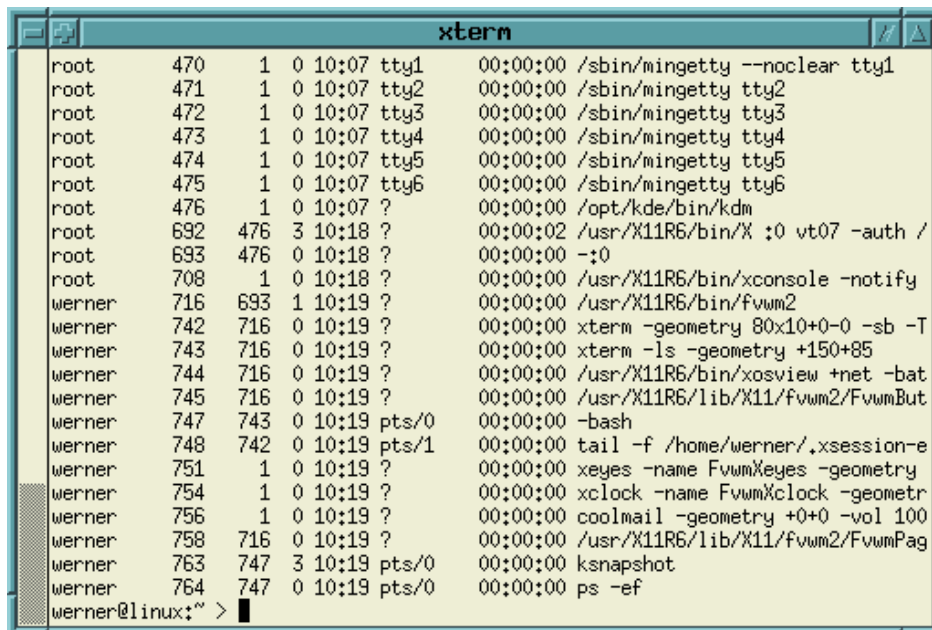


Figure 2-6: An xterm window with a scrollbar

You can also use the `-sb` option for other clients that can run with a scrollbar but don't do so by default. However, this only works with clients for which a scrollbar makes sense. In addition, many applications provide horizontal as well as vertical scrollbars.

The X scrollbar is created by the Athena Scrollbar widget, which moves text according to which pointer button you use and how you use it. The scrollbar has two parts: a *thumb* (the highlighted area within the scrollbar) and the *scroll region*, as seen in Figure 2-6. The thumb moves within the scroll region and displays the position and amount of text currently showing in the window relative to the total amount saved. When an *xterm* window with a scrollbar is first created, the thumb fills the entire scrollbar. As more text is saved, the size of the thumb decreases. The number of lines saved is 64 by default, but an alternative can be specified with either the `-sl` command-line option or the `saveLines` value in a resource file as described in the next chapter.

When the pointer is positioned in the scrollbar, the cursor changes to a two-headed arrow. You can then scroll the text in various ways by clicking or dragging with certain pointer buttons. Table 2-5 summarizes the scrollbar commands. However, the first command may be the only one you need--you can drag the text either up or down using the second pointer button. This command is the simplest and offers the most control over how much scrolling takes place. When you release the button, the window displays the text at that location. This makes it easy to get to the top of the data by pressing the second button, dragging the thumb to the top of the scroll region, and releasing the pointer button.

To get back to the current cursor position, press any key. Either the spacebar or RETURN is a good choice.

Table 2-5: Scrollbar Commands

To move text in this direction	Place pointer on scrollbar and	Notes
Either up or down	Hold down second pointer button and drag thumb	Text follows pointer movement.
Down	Click first pointer button.	Scrolls towards bottom of window (most recently entered text).
Up	Click third pointer button.	Scrolls towards top of window (oldest text).
Either up or down	Click second pointer button	Scrolls to a position in text that corresponds to the pointer's position in scroll region.

Copying and Pasting Text Selections

Using the pointer, you can select text within an *xterm* to copy and paste into the same or another *xterm* window. You don't need to be in a text editor to copy and paste; you can also copy or paste text to and from the command line.

Text copied into memory using the pointer is saved in a global cut buffer, and it also becomes what is known as the PRIMARY text "selection." Both the contents of the cut buffer and the contents of the PRIMARY text selection are available to all clients. When you paste text into an *xterm* window, by default the contents of the PRIMARY selection are pasted. If no text is in the PRIMARY selection, the contents of the cut buffer (called CUT_BUFFER0) are pasted. In most cases, these are identical and you don't have to think about it. However, this mechanism does become important when you want to perform certain customizations, particularly those involving the *xclipboard* client described in Chapter 3.

For now, however, let's just consider the standard methods for copying and pasting text between *xterm* windows.

Selecting text to copy

There are several ways to select (copy) text in an *xterm* window, all using the pointer. You can select a passage of text, or you can select text by individual words or lines. Hypothetically, the selected passage can be of any length. However, the size of the window limits the amount of text you can copy at one time; there can also be problems pasting long selections.

There are two methods for selecting a passage of text. The first is to place the pointer at the beginning of the text, click the first button, and drag the pointer to the end of the text. The second method is to Click the first pointer button at the beginning of the text you want to select and then click the third pointer button at the end of the text.

In either case, the text is highlighted and copied into memory. (Technically speaking, the text is copied into CUT_BUFFER0 and is also made the PRIMARY text selection.)

You can select a single word by placing the pointer anywhere on the word and double-clicking the first button. To select a line, place the pointer anywhere on the line and triple-click the first button.

Each selection replaces the previous contents of CUT_BUFFER0 and the previous PRIMARY text selection--you can make only one selection at a time. (The *xclipboard* client can be used to store multiple text selections.)

Table 2-6 summarizes the text selection methods.

Table 2-6: Button Combinations to Select Text for Copying

To select	Do this
Passage	At the beginning of the selection, hold down the first button; move the pointer to the end of the desired text; and release the button. Or: Click the first button at the start of the selection and the third button at the end of the selection.
Word	Double-click the first button anywhere on the word.
Line	Triple-click the first button anywhere on the line.

To clear the highlighting, move the pointer off the selection and click the first button anywhere else in the window. Note, however, that the text remains in memory until you make another selection; this can be useful if you don't want to paste the text immediately, or if you want to paste it in more than one place.

Extending a selection

Regardless of how you make a selection, you can extend that selection, generally using the third pointer button. The simplest way to extend a selection is to move the pointer to the right or the left to encompass the text you want to add to the selection and click the third pointer button. Your selection now extends from the previous selection to the new endpoint.

Alternatively, you can press and hold down the third pointer button and drag the pointer to extend the selection. Then release the pointer button.

While an extension always begins from the previous selection, the new endpoint doesn't have to be in the same line and it can result in a smaller selection. By moving the pointer up or down, or to the right or left of the last selection, you can select part of a line or add or subtract several lines of text.

If the previous selection was by word or line, when you extend it, the extension is automatically also by word(s) or line(s). Again, there are a few ways of extending a selection made by word or line.

First, continuing to hold the button down after double- or triple-clicking (rather than releasing it) and moving the pointer selects additional text by words or lines at a time. Releasing the button ends the selection.

More commonly you will probably decide to extend the selection after making it (and after releasing the first pointer button). Under these circumstances, you can extend it by moving the pointer and clicking the third button. If you originally selected an entire line

by triple-clicking the first pointer button, moving the pointer to another line and clicking the third button extends the selection to encompass that new line and all lines in between.

As an alternative, you can extend a word or line selection by pressing and holding down the third pointer button, dragging the pointer, and releasing the third button. The extension still increments by word or line as appropriate. Table 2-7 summarizes the button combinations for extending a text selection.

Table 2-7: Button Combinations to Extend a Text Selection

To select	Do this
By passage	Move the pointer to the place in the text to which you want the selection to extend and click the third button. Or: Hold down the third button, move the pointer to the end of the text to be included, and release the button.
By word	Move the pointer to the last word you want to include in the selection and click the third button anywhere on the word. Or: Hold down the third button, drag the pointer onto the last word you want to include, and release the button. Or: After double-clicking the first button anywhere on the word to select it, continue to hold the button down, and drag the pointer. When you've included the words you want, release the button.
By line	Move the pointer to the last line you want the selection to include and click the third button anywhere on the line. Or: Hold down the third button, drag the pointer onto the last line you want to include, and release the button. Or: After triple-clicking the first button anywhere on the line to select it, continue to hold the button down and drag the pointer up or down. When you've included the lines you want, release the button.

To select text that fills more than one screen, select the first screenful. Use the scrollbar to view the additional text. Then use the third pointer button to extend the selection. The original selection does not need to be in view; clicking the third button extends the selection to the point you choose.

Pasting text selections

Clicking the second pointer button inserts the text from the PRIMARY selection (or CUT_BUFFER0, if the PRIMARY selection is empty) as if it were keyboard input. You can copy data from one *xterm* window to another by selecting the data in one window, moving the pointer to another window, and clicking the second button.

You can paste text either into an open file or at a command-line prompt. To paste text into an open file, click the second button within the text-editor window containing the file. The text from the memory area is inserted at the location of the editor's cursor. You can paste the same text as often as you like. The contents of the PRIMARY selection remain the same until you make another selection.

To paste text at a command-line prompt, click the second button anywhere within the window to place the text on the current command line. (Note that the window scrolls to the bottom on input.) This is a quick and easy way to copy a command from one window to another.

Note that you can paste text into a window when click-to-type focus is in effect, even if the window does not have the input focus; the focus is moved to the window once you've clicked on it to insert the text.

Running a Program in a Temporary *xterm* Window

Normally, when you start up an *xterm* window, it automatically runs another instance of your shell. If you want to create an *xterm* window that executes some other program and goes away when that program terminates, you can do so with the *xterm -e* option:

```
| xterm -e command [arguments]
```

For example, if you want to look at the file *temp* in a window that will disappear when you quit out of the file, you can use the *-e* option to run the *less* program:

```
| xterm -e less temp
```

The titlebar of the *xterm* window displays the name of the command you are running (unless you've also specified an alternative string using *-title*).

When you are also using other *xterm* options, the *-e* option must appear last because everything after *-e* is assumed to be part of the command to be executed.

The *xterm* Menus

xterm has four menus, each serving a different purpose. These menus are displayed by placing the pointer in the window proper and pressing the CTRL key and a pointer button simultaneously. (The exact key and button combinations are described in subsequent sections with each menu.)

The following menus are available:

- Main Options menu
- VT Options menu
- VT Fonts menu
- Tek Options menu

The *xterm* menus are divided into sections separated by horizontal lines. The menu options include various modes that can be toggled and commands to be executed. A check mark appears next to a mode that is currently active. Selecting a mode toggles its state.

Some options are mutually exclusive. For example, the menu items at the top of the VT Fonts menu change the size of the font in which text is displayed in the *xterm* window. Only one font size can be active at a time; to turn one size off, you must activate another.

Most of the mode entries can also be set by command-line options when invoking *xterm* as described in this chapter, or by entries in a resource startup file as described in Chapter 12. The various modes on the menus are helpful if you've set (or failed to set) a particular mode and then decide you want a different option.

The sections below the modes portion of each menu contain various commands. Selecting one of these commands performs the indicated function. Many of these functions can only be invoked from the *xterm* menus. However, some functions can be invoked in other ways; for example, from an *fvwm* menu, on the command line, or by a sequence of keystrokes.

When you display an *xterm* menu, the pointer becomes the arrow pointer and initially appears in the menu's title. Once the menu appears, you can release the CTRL key. The menu remains visible as long as you hold down the pointer button. To select a menu item, move the pointer to that item and release the pointer button. Any selection you make from a menu applies only to that *xterm* window.

The Main Options menu

Bring up the Main Options menu by moving the pointer inside your *xterm* window and holding down the CTRL key while you press and hold the first pointer button. When the menu appears, you can release the CTRL key.

The menu is shown in Figure 2-7. It allows you to set certain modes and to send signals (such as SIGHUP) that affect the *xterm* process.

```

Main Options
Secure Keyboard
Allow SendEvents
Print Window
Redraw Window
-----
8-Bit Controls
Backarrow Key (BS/DEL)
✓ Alt/NumLock Modifiers
Meta Sends Escape
Delete is DEL
✓ Old Function-Keys
HP Function-Keys
SCO Function-Keys
Sun Function-Keys
VT220 Keyboard
-----
Send STOP Signal
Send CONT Signal
Send INT Signal
Send HUP Signal
Send TERM Signal
Send KILL Signal
-----
Quit
```

Figure 2-7: Main Options menu

To select a menu item, move the menu pointer to that item and release the first button. The menu is divided into four sections. After you have selected a mode in the top section (Secure Keyboard or Allow SendEvents), a check mark appears before the item to remind you that it is active. In addition, Allow SendEvents can be set by an entry in a resource

startup file such as *.Xresources*. The menu selections enable you to change your mind once *xterm* is running.

The Secure Keyboard mode toggle is intended to help counteract one of the security weaknesses of X. Generally, when you press a keyboard key or move the pointer, the X server generates a packet of information, known as an *event*, that is available for other clients to interpret. Because events such as the keys you type in an *xterm* window are made available via the server to other clients, an adept system hacker could access this information.

A serious breach of security could easily occur, for instance, if someone were able to find out a user's password or the *root* password. Enabling Secure Keyboard mode causes all user input to be directed *only* to the *xterm* window itself. If your environment might be vulnerable, you can enable Secure Keyboard mode before typing a password or other important information and then disable it again.

When you enable Secure Keyboard mode, the foreground and background colors of the *xterm* window are exchanged. When you disable Secure Keyboard mode, the colors are switched back.

Only one X client at a time can secure the keyboard. Thus, if you have enabled Secure Keyboard mode in one *xterm*, you cannot enable it in another *xterm* until you disable it in the first. If Secure Keyboard mode is not available when you request it, the colors are not switched and a bell sounds. If you request Secure Keyboard mode and are not refused but the colors are *not* exchanged, be careful: you are not in Secure Keyboard mode.

Due to an X protocol limitation, Secure Keyboard is disabled automatically if you iconify the *xterm* window. When the mode is disabled, the colors are switched back and the bell sounds.

Unlike Secure Keyboard, which protects you from a security risk, the Allow SendEvents option opens up a potential security hole. Toggling Allow SendEvents permits the *xterm* to receive artificially generated events from the X server—in other words, someone at a remote terminal could send commands to your *xterm*. Thus we don't recommend using this option.

The Print Window option, not too surprisingly, prints the window, and Redraw Window redraws the contents of the window. As an alternative, you can redraw the entire screen using the *xrefresh* client.

The second section of the Main Options menu has a set of toggles that control how various keys work. The Backarrow Key option toggles the behavior of the backspace key between sending a backspace (BS) and sending a delete (DEL). Toggling this and the Delete is DEL key can be used to switch the default behavior of these keys. The function key entries toggle the behavior of the function keys. Only one of these options can be selected at a time. If the option 8-Bit Controls is set, control sequences are sent as 8-bit characters rather than escape sequences. This option may be unavailable, as shown in Figure 2-7. Alt/NumLock Modifiers controls the use of the Alt and NumLock keys for Sun/PC and VT220 keyboard extensions.

The third section includes several commands that send a signal that is intended to affect the *xterm* process: suspend it (Send STOP Signal), terminate it (Send TERM Signal), etc. Note that most of these commands are equivalent to common keystroke commands, which

are generally simpler to invoke. For example, in most terminal setups CTRL-C can be used to interrupt a process. This is generally simpler than using the Send INT Signal menu command, which performs the same function.

Similarly, you can suspend a process by typing CTRL-Z and start the process again by typing CTRL-Y, rather than using the Send STOP Signal and Send CONT Signal menu commands.

Four of the commands (Send HUP Signal, Send TERM Signal, Send KILL Signal, and Quit) send signals that are intended to terminate the *xterm* window. In most cases you can probably end an *xterm* process simply by typing some sequence (such as CTRL-D or *exit*) in the window or from the window operations menu. Of course these menu items may be very helpful if the more conventional ways of killing the window fail. See the *signal* manpage for more information on what each signal does.

The Quit command sends a SIGHUP to the process group of the process running under *xterm*, usually the shell. (The Send HUP Signal command sends the same signal.) This kills the *xterm* process, and the window disappears from the screen. Quit is in a section by itself, so it's easier to point at. Sending a SIGHUP with Quit is a cleaner way to shut down an application than sending a SIGKILL with the Send KILL Signal.

The VT Options menu

To bring up the VT Options menu, move the pointer to the *xterm* window, hold down the CTRL key, and then press and hold down the second pointer button. The menu shown in Figure 2-8 then appears.

VT Options

- ✓ **Enable Scrollbar**
 - Enable Jump Scroll
 - Enable Reverse Video
 - ✓ **Enable Auto Wraparound**
 - Enable Reverse Wraparound
 - Enable Auto Linefeed
 - Enable Application Cursor Keys
 - Enable Application Keypad
 - ✓ **Scroll to Bottom on Key Press**
 - ✓ **Scroll to Bottom on Tty Output**
 - Allow 80/132 Column Switching
 - Enable Curses Emulation
 - Enable Visual Bell
 - Enable Margin Bell
 - Enable Blinking Cursor
 - ✓ **Enable Alternate Screen Switching**
 - Enable Active Icon
-
- Do Soft Reset**
 - Do Full Reset**
 - Reset and Clear Saved Lines**
-
- Show Tek Window**
 - Switch to Tek Mode**
 - Hide VT Window**
 - Show Alternate Screen**

Figure 2-8: VT Options menu

The VT Options menu provides many VT102 setup functions, which can be used to specify certain characteristics of the *xterm* window. Some of these mode settings are analogous to those available in a real VT102's setup mode; others, such as *scrollbar*, are *xterm*-only modes.

The VT Options menu items allow you to reset several characteristics or modes at once, select the Tektronix window to accept input, hide the VT window, etc.

Check marks indicate the active modes. For example, Enable Scrollbar, Auto Wraparound, Scroll to Bottom on Key Press, Scroll to Bottom on Tty Output,* and Enable Alternate Screen Switching are active in the VT Options menu displayed in Figure 2-8.

These are the only modes active by default. To turn off one of these modes, move the menu pointer to that mode and release the second button. Most of these modes can also be set by command-line options when invoking *xterm* or by entries in a resource startup file

* This mode indicates that if you are using the scrollbar and the window receives output (or a key is pressed, if *stty echo* is enabled), the window scrolls forward so that the cursor is at the current line. (You can use the menu to toggle off this mode but it is generally desirable to leave it on.)

like *.Xresources*. The menu selections allow you to change your mind once *xterm* is running.

The toggle Allow 80/132 Column Switching warrants a little more explanation. This mode allows *xterm* to recognize the DECCOLM escape sequence, which switches the terminal between 80- and 132-column mode. The DECCOLM escape sequence can be included in a program (such as a spreadsheet) to allow the program to display in 132-column format. This mode is off by default.

The VT Options menu commands (in the second and third partitions of the menu) perform two sets of functions, neither of which can be performed from the command line or a resource definition file. The commands Do Soft Reset and Do Full Reset reset some of the modes on the menu to their initial states. Reset and Clear Saved Lines does a Full Reset and also clears the saved lines that have scrolled off the window. (See the *xterm* manpage for more information.)

The Show Tek Window, Switch to Tek Mode, and Hide VT Window menu items allow

Tek mode supports a relatively old graphics standard that requires emulator support of the Tektronix Tek4010 graphics display. Unlike the DEC VT100 terminal, which only allowed characters, the Tektronix 4010 displayed its contents as graphics. It considered everything as a group of dots to be plotted.

you to manipulate the Tektronix and VT102 windows.

The Show Tek Window command displays the Tek window and its contents without making it the active window (you can't send input to it). Use the Switch to Tek Mode command to display a Tektronix window and make it the active window. When you select Switch to Tek Mode, the Show Tek Window command is automatically enabled, since the Tek window is displayed. Both of these commands are toggles. If Show Tek Window is active and you toggle it off, the Tek window becomes hidden. If both Switch to Tek Mode and Show Tek Window are active, toggling off either one of them switches the *xterm* back to VT mode. The Hide VT Window command hides the VT102 window but does not destroy it or its contents. You can restore it and make it the active window by choosing Select VT Mode from the Tek Options menu.

The VT Fonts menu

To bring up the VT Fonts menu, move the pointer inside the *xterm* window. Press and hold down the CTRL key on the keyboard and press the third (usually the right) pointer button. This menu allows you to change the display font of an *xterm* window while the window is running, a powerful and useful capability. The VT Fonts menu is shown below in Figure 2-9.

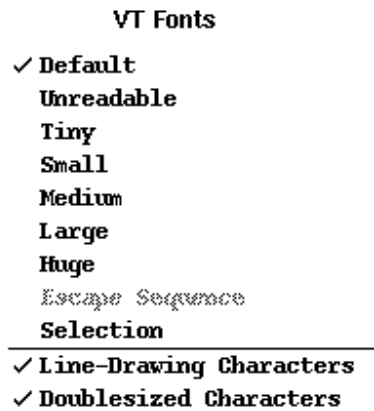


Figure 2-9: VT Fonts menu

If you have not toggled any items on this menu, a check mark appears before the Default mode setting. The default is the font specified when the *xterm* window was run. This font could have been specified on the *xterm* command line or in a resource file. Whatever the case, this font remains the default for the duration of the current *xterm* process.

The items Default, Unreadable, Tiny, Small, Medium, Large, and Huge can be toggled to set the font displayed in the *xterm* window. The font can be changed any number of times to accommodate a variety of uses. You might choose to use a large font for editing a file (chances are you've chosen a large enough default font, though). You could then change to a smaller font while a process is running since you don't need to be reading or typing in that *xterm*. Changing the font also changes the size of the window.

You can also change the font size in an *xterm* window using the keyboard shortcuts Shift-KP+ and Shift-KP-. The combination of the Shift key and the keypad + key makes the font larger, while the Shift key and the keypad – key makes it smaller.

There are also default settings for the Unreadable, Tiny, Small, Medium, Large, and Huge fonts. Bring up the VT Fonts menu and toggle some of these fonts to see what they look like. If you select the Unreadable font, your *xterm* window becomes very tiny, almost the size of some application icons. Though you cannot read the actual text in a window this size, the window is still active and you can observe if additional output, albeit minuscule, is displayed. An *xterm* window displaying text in such a small font can, in effect, serve as an *active icon*.

You can specify your own Unreadable, Tiny, Small, Medium, Large, and Huge fonts in a resource file. The corresponding resource names are *font1*, *font2*, *font3*, *font4*, *font5*, and *font6*. See Chapter 12 for instructions on how to set resource variables and Chapter 13 for more information about available fonts.

Following the font selections, the VT Fonts menu offers two other possible selections: Escape Sequence and Selection. When you first run an *xterm* window, these options appear on the VT Fonts menu but they are not available. See Chapter 13 for information on using these options.

Finally, below the line are the two remaining options: Line-Drawing Characters and Doublesized Characters. Toggling Line-Drawing Characters indicates whether or not the current font has line-drawing characters available that can be drawn directly. Toggle Doublesized Characters on to tell *xterm* that it can use font scaling to draw doublesize characters, rather than simulating them with normal characters separated by spaces.

The Tek Options menu

To display the Tek Options menu, you must be in the Tektronix window (by selecting Display Tek Window from the VT Options menu). Then press and hold down the CTRL key on the keyboard and press the second pointer button. We don't show the Tek Options menu here because it's almost never used nowadays, but it controls certain modes and functions of the Tektronix window, including setting the size of the characters in the Tektronix window and hiding the window again.