Chapter 3

An Overview of the System

With FreeBSD installed on your computer, it's time to begin using the OS. This chapter provides a quick overview of FreeBSD, including the boot process, logging in, running programs, starting and using X, logging off, and shutting down. If you use FreeBSD as a desktop workstation, chances are you'll need to know how to perform most of these tasks on a fairly regular basis. FreeBSD systems often go for days or even months without rebooting, though, so the booting and shutdown actions, although important, may be infrequent. If your FreeBSD system is a server, you might not run X on it, but the remaining topics are important.

Booting the Computer

In one sense, booting a FreeBSD computer is a simple matter—you turn on the power and wait for the system to come up. You may need to deal with an option or two, though, and the boot process provides a great deal of information that you may find useful later, particularly if you need to debug a problem. This section therefore describes the boot process from a practical perspective.

Selecting FreeBSD

The early stages of an *x*86 computer's boot process are controlled by the Basic Input/Output System (BIOS) installed in the system. Once the BIOS finishes its self-tests and other startup procedures, it hands control over to a boot loader that's stored on disk—typically the hard disk when booting an installed OS, although floppies, CD-ROMs, and other removable media devices can contain boot code as well.

If you installed FreeBSD alongside another OS, chances are you'll see a boot prompt that resembles the following (the FreeBSD *Stage 0* boot prompt):

```
F1   DOS
F2   FreeBSD

Default: F2
```

FreeBSD labels Windows partitions *DOS*, so don't be concerned if this is what you see and your non-FreeBSD partition holds Windows. Press the key (F1, F2, or some other key if there are more than two options) that corresponds to the OS you intend to boot—FreeBSD in the case of this example. The default value corresponds to the OS you booted last. If you regularly boot one OS, you can wait for the system to boot this default OS, but for a quicker boot, press the appropriate function key.

Note *If you've installed a boot loader other than FreeBSD's default, you see it rather than the FreeBSD boot loader. Chapter 4 includes a discussion of third-party boot loaders.*

Once FreeBSD starts to boot, it begins displaying status messages. The first of these relates to the bootstrap loader, which is a secondary boot loader (also known as the *Stage 1* and *Stage 2* boot loaders, in FreeBSD parlance; the two are distinct programs, but function as one). The Stage 0 boot loader is very small and simple; it merely redirects the boot process to the appropriate partition. The Stage 1/Stage 2 boot loader is more complex and more FreeBSD-specific. It displays a message similar to the following:

```
Hit [Enter] to boot immediately, or any other key for command prompt.
Booting [/boot/kernel/kernel] in 9 seconds...
```

The time in the second line counts down. Ordinarily, you can wait for it to time out to boot FreeBSD, or press ENTER to speed up the process slightly. In some rare circumstances, though, you may need to load special kernel modules (that is, hardware drivers) or otherwise configure the kernel. If so, press any other key and the bootstrap loader allows you to enter commands. Some of them are described in Chapter 4.

You may enter any special parameters for the bootstrap loader, or let it time out, or press ENTER to have it move on without special options. Once this happens, the bootstrap loader loads the FreeBSD kernel into memory, from the file `/boot/kernel/kernel` or from another file that you specify as an option. The kernel then displays status information and runs FreeBSD startup scripts.

Interpreting Boot Messages

Unlike Windows, which displays little in the way of startup messages, FreeBSD is quite verbose as it starts up. In fact, it's so verbose that much of the information it presents scrolls off the screen before you have a chance to read it. Fortunately, the kernel messages are stored for a time after you boot, and you can view them by typing **`dmesg`**. You may want to redirect these messages into a file for perusal with a text editor (as in **`dmesg > dmesg.txt`**) or pipe the result through a text viewer such as `less`, as in **`dmesg | less`**. You won't be able to do these things until after you've logged in, though, as described later in this chapter in the section "Logging Into an Account."

Note *Redirection and piping, which are the techniques used in the preceding commands to send the output of `dmesg` to a file or through a text viewer, are common methods of getting programs to work together or to capture program output in FreeBSD. These topics are covered in Chapter 5.*

Whether you're quickly scanning the boot messages as they're displayed or studying them in more depth after you log in, these messages can be a useful source of information about your system. They begin with a series of copyright messages and information on the version of FreeBSD you have installed. Subsequent lines identify your CPU, including its make, model, and clock speed. The system also reports the amount of memory you have installed. This includes both the *real memory* (the amount on the SIMMs, DIMMs, or other memory chips you've installed in the computer) and the *available memory* (the real memory minus that used by the kernel, BIOS caches, and so on).

The bulk of the information that the kernel displays consists of reports on various hardware devices. These reports typically begin with a device name (which usually bears little resemblance to any name you might use for the device), a colon (`:`), and a more descriptive expansion or report of details. Some devices consume more than one line. For instance, the following is a report that's likely to appear very early:

```
npx0: <math processor> on motherboard
npx0: INT 16 interface
```

This identifies the computer as supporting a math coprocessor, which is a component that performs floating-point arithmetic. All CPUs since the 486DX series include a math coprocessor as part of the main CPU, but 386, 486SX, and some NexGen CPUs lacked this feature. Such systems often supported math coprocessors as add-on chips that fit into motherboard sockets, hence FreeBSD's claim that the math coprocessor is on the motherboard.

Other devices that you're likely to see mentioned include:

- **PCI and ISA busses** The `pcib0`, `pci0`, `isab0`, and `isa0` devices relate to the PCI and ISA *busses*—that is, the slots in which you insert expansion cards.
- **ATA controllers** The *Advanced Technology Attachment (ATA)* interface is another name for the Enhanced Integrated Drive Electronics (EIDE) hard disk interface. Modern computers are likely to have three ATA devices: `atapci0` (the ATA controller, which is built into modern motherboards) and the devices associated with the two ATA busses, `ata0` and `ata1`. Some systems may have more or fewer ATA devices than this, though.
- **SCSI host adapters** If your system has a SCSI host adapter, you're likely to see a device for it. The name varies depending upon the type of chipset used on your SCSI adapter.
- **USB controllers and devices** You may see several devices whose names begin with *u* that relate to USB support. Most systems have a `uhci0` or `ohci0` device that corresponds to the USB controller itself, a `usb0` device that serves as an abstraction of the USB controller, and a `uhub0` device that grants access to the USB ports on the computer. You may have additional USB devices, such as `umass0` for a mass-storage device such as a Zip disk, or `ums0` for a mouse.
- **Network devices** The names of your network devices, if any, vary depending upon the chipsets used. There's usually a core device, whose description includes the name of the chipset used, and several additional devices related to specific component parts of the network device.
- **Keyboard and mouse** If you're using a standard keyboard, FreeBSD reports it as `atkbdc0` and `atkbd0`. The mouse may be `psm0` (for a PS/2 mouse), `ums0` (for a USB mouse), or something else, depending on its interface type.

- **Parallel and serial devices** The parallel port (`ppc0`) may support several subdevices for printers (`lpt0`), generic parallel-port input/output (`ppi0`), and networking (`plip0`). The serial ports (`sio0` and `sio1`, typically) might conceivably show a Point-to-Point Protocol (PPP) device (`ppp0`), but this normally appears only after you've explicitly started a PPP network link.
- **Console** FreeBSD uses a special device to access the console (`sc0`)—normally the keyboard and monitor attached directly to the computer.
- **Video** Most systems support a `vga0` device for the video card.
- **Disk devices** The floppy disk (`fd0` and `fdc0`), EIDE hard disks (`ad0` and up), SCSI hard disks (`da0` and up), and CD-ROM drive (`acd0` for EIDE disks) all have their own devices. You may see an error message about an inability to find the device's capacity for some removable drives, such as Zip drives, unless you boot with a disk in the drive. Don't be concerned about such messages.

If you notice error messages related to any of these devices, it may indicate that FreeBSD has not detected the device correctly or it isn't working. If some important device is missing, it's possible that it's something for which no FreeBSD driver exists, or the device may be damaged or inoperable. Some devices may not show up immediately. For instance, external devices that aren't turned on won't appear. You may also see some devices labeled as `unknown`. These are probably devices that FreeBSD found but could not configure correctly. This situation sometimes happens with plug-in cards for which no FreeBSD drivers exist.

Understanding Startup Scripts

After starting the kernel, the FreeBSD boot process runs a program called `init`. This program controls the startup process once the kernel is loaded and running. It does this by running various startup scripts in the `/etc` directory, starting with `/etc/rc`. These scripts control the startup process, launching servers and other programs that must run in the background. Many of these actions produce further output on the screen, although this output is not stored for retrieval via `dmesg`, as are the kernel startup messages.

Some of the tasks performed by `init` and its startup scripts include:

- **Filesystem checks** When FreeBSD starts, it checks partitions to be sure partitions are *clean*—that is, that they were properly unmounted. If they're not clean, FreeBSD runs the `fsck` program, which checks the partitions and corrects any problems that might have resulted from a crash or other unclean shutdown.
- **Mounting partitions** FreeBSD mounts its partitions under direction of the `/etc/fstab` file. This file is described further in Chapter 7. For now, know that it contains information on partitions and mount points, as you defined them when you installed FreeBSD.
- **Miscellaneous configuration** FreeBSD must perform a large number of miscellaneous startup tasks to do things such as configure its network interface, enable PC cards on laptop computers, and initialize serial ports.

- **Running servers and other programs** Some programs must run continuously to do any good. These include, but are not limited to, most servers. They're normally started automatically at boot time.

Note *Most servers run as* daemons. *This word derives from the Greek, and means "helper." In FreeBSD (and UNIX generally), daemons typically run unattended in the background, in order to provide some necessary service. Not all daemons are servers, but many are. The similarity of this word to the English word* demon *is what gives rise to the horned red FreeBSD mascot.*

- **Enabling logins** In order to log into a FreeBSD system, that computer must be running some form of login program—a program that accepts a username and password and grants the user access to the system in response. Some login programs are network servers, but others run locally. FreeBSD starts the latter class via entries in the `/etc/ttys` file.

Chapter 6 describes the system startup scripts in more detail. For the moment, it's important that you know that FreeBSD starts many programs during its startup process, including those that enable you to log onto the system. As described in Chapter 6, you can modify the FreeBSD startup process to add new programs or delete those you don't need.

Logging into an Account

Once the system boots, FreeBSD displays a login prompt of one sort or another. Logging in involves entering your username and password. Before you do this, you should know something of the different login types—text mode vs. GUI, and user vs. `root` logins. You should also know what to expect once you've logged in, and how to acquire `root` privileges once you've logged into a user account.

Text-Mode and GUI Logins

The default configuration for FreeBSD is to provide a text-mode login. This means that when you boot the computer, you'll see a text-mode prompt that resembles the following:

```
login:
```

This is FreeBSD's way of asking for your username. Type it, and then press ENTER. FreeBSD responds with another prompt, immediately below the first:

```
Password:
```

Enter your password at this prompt, and then press ENTER again. As a security measure, you won't see your password echoed to the screen. (The security benefit is that nobody lurking nearby sees it, either.) If you enter the correct password, FreeBSD displays a screenful of information about where to find security advisories, FreeBSD documentation, and so on. At the bottom of the screen you'll see a *command prompt*, which is probably a single dollar sign (`$`). The `root` account's prompt is a pound sign (`#`), though, to help you identify when you're logged in as `root`. Depending upon your account's configuration, you may see additional information just prior to the prompt, such as your username or the name of your shell.

A text-mode login such as this allows you to run text-based programs and use text-mode commands, as described in the upcoming section, "A Summary of Text-Mode Commands." Sometimes you may want to run multiple text-mode programs. Two common ways to do this are

- **Virtual consoles** FreeBSD supports a feature known as *virtual consoles*, which enables you to log in several times. After you've logged in once, press F2, or any other function key up to F8. You'll see another `login:` prompt, and you can log in a second time. Press F1 to return to the first virtual console. You can then run different programs in the different screens. You can even log in as different users in the different virtual consoles. (Once you start running X, as described later in this chapter in the section "Starting X," you must press ALT along with the function key to switch to another virtual console. X itself runs in the 9th virtual console, accessed via F9.)
- **Background processing** You can run one program in the background, meaning that it no longer receives your keyboard input. To do this, append an ampersand (&) to its command name when you run the program, as in **`numbercrunch &`**. Alternatively, you can suspend a program's operation by pressing CTRL-Z when it's running; this stops the program without killing it and returns you to a command prompt. This step is particularly useful if you want to temporarily exit from a program and then return to it; typing **`fg`** gets you back into the program, and typing **`bg`** sets it running in the background, as if it had been started with an ampersand to begin with.

If you log in using text mode, you won't be able to run GUI programs without first starting the X Window System (or X for short), as described in the upcoming section, "Starting X." You can configure a FreeBSD system to automatically start X when the system boots. Such a system uses the *X Display Manager (XDM)* program, or a similar tool, to provide a GUI login screen, as shown in Figure 3-1. When you type your username and password in this screen, FreeBSD automatically starts your X environment. You may prefer this configuration if you do most of your work in X. Properly configuring such a setup requires starting the XDM program in one of the startup scripts, and you may need to adjust users' individualized X configurations, as well. Chapter 21 describes how to configure XDM or similar programs, and Chapter 6 covers users' login control files, including those for use with XDM.

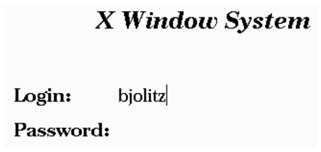


Figure 3-1. *A GUI login runs in X and logs users directly into their X-based environments.*

User vs. root Accounts

As described in Chapter 2, you're likely to use at least two FreeBSD accounts: an ordinary user account and the `root` account. You employ a user account for day-to-day user tasks, such as reading e-mail, browsing the Web, using a spreadsheet, and manipulating graphics files. The `root` account is the account used by the system administrator, aka the *superuser*. This account provides more-or-less unlimited control of the computer; `root` can read, write, move, or delete any file on the computer. For most tasks, you should use an ordinary user account; reserve `root` logins for the times when you need to administer the system.

Note *In this book, commands you type are shown after a prompt. In most cases, this prompt is either a dollar sign (`$`) or a pound sign (`#`), signifying the command's use from a user account or as `root`, respectively. This mirrors the default prompts displayed on the screen when you're logged in as an ordinary user or as `root`.*

It's not uncommon to find that you've logged in as an ordinary user, but need superuser privileges for some short action. This problem can be overcome in several ways. One is to use FreeBSD's virtual consoles to log in as `root`, perform whatever action you need to do, and log out again. Such a procedure can be awkward if you're running in X and want to run a program that requires `root` privileges alongside other programs, though. Even when running in text mode, there's a method that's often simpler: `su`.

The `su` program name stands for *substitute user*. It's a method of changing the identity of a user who's already logged in. It can take several options and parameters:

```
su [-] [-flm] [-c class] [username [args]]
```

Note *FreeBSD documentation often lists command syntax as in this example. Square brackets indicate a parameter that's optional. Single-character options within brackets can be added or deleted individually (as in `-flm` in this example; you can use just `-f`, just `-l`, just `-m`, or any combination you like). Italic indicates variables. Ellipses (not shown here) indicate that an option may be repeated. A vertical bar (`|`, also not shown here) separates two alternatives that aren't used together.*

The simplest use of su is to type it alone, as in **su**. If the user who types it is in the wheel group, su prompts for the root password. If the user types this password correctly, the user acquires superuser privileges. If you enter a username on the su command line, you acquire that user's identity. In either case, typing **exit** ends the identity change. You can further modify su's behavior by adding various optional parameters:

- **-l or a single dash (-)** Ordinarily, su creates an environment that's not quite identical to a normal login environment. The -l option (or a simple dash by itself) discards most of the user's current environment and creates a new one that should match the target user's normal environment.
- **-f** If the target user uses csh as a shell, this option causes su to not read the csh configuration file, .cshrc. You can use this to make the target environment *less* like the target user's normal environment.
- **-m** This option changes your identity without changing your environment at all. For instance, you'll continue to use your normal command shell, even if the target user normally uses a different one.
- **-c *class*** FreeBSD supports login *classes*, which are similar to groups. You can specify a class whose settings you want to use.
- ***args*** If you want to pass arguments (control parameters) to the target user's command prompt shell, you can do so by placing them at the end of the su command.

You'll find that su is an extremely useful tool for administering FreeBSD. Using su, you can log in as an ordinary user and run privileged programs when necessary by first typing **su**. This procedure works both from text-mode logins and from GUI logins once you've started a window in which you can enter commands.

You can log in directly as root at the console, but as a security measure, FreeBSD doesn't accept remote logins via Telnet as root. This restriction means that a remote attacker who knows the root password must also know another user's password to invade the system—a would-be attacker must log in as an ordinary user and then use **su** to acquire root privileges. Of course, the same is true of authorized administrators, which may be an inconvenience. Nonetheless, the increased security is worth this inconvenience. Indeed, even at the console, it's best to not log in directly as root. When you use su, FreeBSD logs the fact, including the user who issued the su command. This logging can help you track who's doing administrative tasks, which may be important if you find that somebody's abusing superuser power.

A Summary of Text-Mode Commands

Because FreeBSD defaults to starting up in text mode, this chapter's first real description of FreeBSD tools focuses on text-mode operations. This topic begins with a description of *shells*—the programs that accept text-mode commands. When running programs from a shell, it's important that you understand a few common standards, such as how the shell interprets filenames and what types of programs you can run. This section then proceeds to cover a few of the more common commands and shell procedures.

Tip *If you're used to GUI-oriented OSs such as Windows or Mac OS, you might be tempted to skim this material as quickly as possible. Many FreeBSD administrative tasks are much easier to perform in text mode than in a GUI, though. Although text mode can be intimidating at first, it's very powerful—more so in FreeBSD than in Windows. Thus, learning to use text-mode commands greatly enhances your ability to handle a FreeBSD system.*

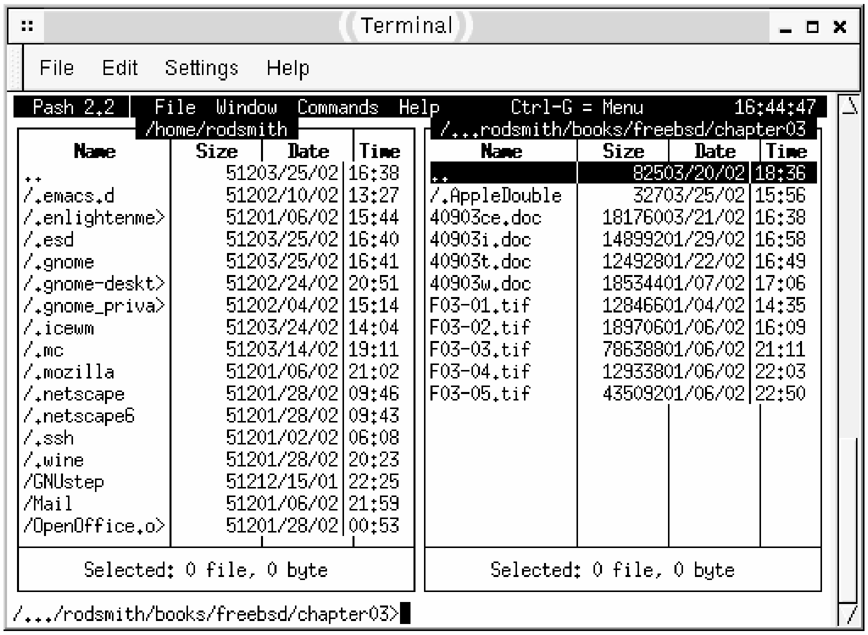
Understanding Shells

When you log into a FreeBSD system in text mode, FreeBSD launches a program—your shell. As described in Chapter 2, the name of your shell is associated with your account. In principle, FreeBSD can use any program as a shell, but by convention only programs designed as shells are used in this capacity for login accounts. The default shell for FreeBSD is `sh`, but others are available. Common choices include `csh` (implemented by `tcsh` in FreeBSD, so these two are equivalent), `bash`, `ksh`, and `zsh`. These shells are ordinary FreeBSD programs. Some reside in `/bin`, but others live in `/usr/local/bin`. For the most part, all these shells operate in the same way, although they differ in their advanced features and in a few less advanced ways. For instance, the default prompts presented by these shells differ, and they use different configuration files; `sh` uses the `.shrc` file in the user's home directory for configuration, whereas `tcsh` uses `.tcshrc` or `.cshrc`, and `bash` uses `.bashrc`. The precise format of these files differs, too, so you can't simply rename, say, `.shrc` to `.tcshrc` and expect it to work. Chapter 6 briefly describes some of these files' formats.

In addition to the traditional UNIX shells, FreeBSD supports some nontraditional shells, such as `mudsh`, which resembles old text-mode adventure games such as Zork in its built-in commands and prompts; and `pash`, which provides a text-mode file browser interface similar to Norton Commander for DOS, as shown in Figure 3-2.

If you want to experiment with various shells, install them using the FreeBSD `sysinstall` utility's `Configure` option, as described in Chapter 11. There's a package area devoted to shells, so you can browse their descriptions and install any that sound interesting. You can then run different shells by typing their names. For instance, if you're in `sh` as the default login shell but you want to try `tcsh`, type **`tcsh`** to launch it. If you decide you want to change your default shell to the one you're trying, you can do so by modifying your account configuration, as described in Chapter 10.

All shells support a combination of built-in and external commands. Most commands you use directly from the shell are external. Most, but not all, of the shell's internal commands are interesting when it comes to scripting: You can write a short program in the shell's



```
 Pash 2.2 | File  Window  Commands  Help        Ctrl-G = Menu        16:44:47
            /home/rodsmith                   /...rodsmith/books/freebsd/chapter03
      Name          Size | Date   | Time         Name        Size  | Date   | Time
 ..                51203/25/02|16:38      ..                 82503/20/02|18:36
 /.emacs.d         51202/10/02|13:27      /.AppleDouble      32703/25/02|15:56
 /.enlightenme>    51201/06/02|15:44      40903ce.doc    18176003/21/02|16:38
 /.esd             51203/25/02|16:40      40903i.doc     14899201/29/02|16:58
 /.gnome           51203/25/02|16:41      40903t.doc     12492801/22/02|16:49
 /.gnome-deskt>    51202/24/02|20:51      40903w.doc     18534401/07/02|17:06
 /.gnome_priva>    51202/04/02|15:14      F03-01.tif     12846601/04/02|14:35
 /.icewm           51203/24/02|14:04      F03-02.tif     18970601/06/02|16:09
 /.mc              51203/14/02|19:11      F03-03.tif     78638801/06/02|21:11
 /.mozilla         51201/06/02|21:02      F03-04.tif     12933801/06/02|22:03
 /.netscape        51201/28/02|09:46      F03-05.tif     43509201/06/02|22:50
 /.netscape6       51201/28/02|09:43
 /.ssh             51201/02/02|06:08
 /.wine            51201/28/02|20:23
 /GNUstep          51212/15/01|22:25
 /Mail             51201/06/02|21:59
 /OpenOffice.o>    51201/28/02|00:53

      Selected: 0 file, 0 byte                Selected: 0 file, 0 byte

/.../rodsmith/books/freebsd/chapter03>
```

Figure 3-2. *Text-mode shells can be run in text mode or in windows from within X, as shown here.*

command language. These scripts can launch other programs, operate on files, and so on. Chapter 31 covers creating such scripts.

Some shells, including `tcsh` and `bash`, support a useful feature known as *command completion*: If you type a partial command and then press TAB, the shell attempts to complete the command. The shell can do two things in response, depending upon what you've typed:

- If you've typed only part of a program name, the shell tries to find a program that matches what you've typed. For instance, if you type **`chm`** and then press TAB, chances are the system will respond by filling out your command as **`chmod`**.
- If you've typed a complete program name and part of a filename to be given to the program as a parameter, the shell tries to locate the file that matches what you've typed so far, much as it tries to find a matching program file in the first case.

In both cases, if the shell finds multiple matches, it either beeps or displays all the possible matches. In some cases, pressing TAB again displays all the matches, if the shell only beeps. If it beeps again, this means there's no match to what you've typed so far.

Note that command completion can't fill out options you give to commands, aside from other files' names. You'll have to know enough about the commands you use to specify the options you want.

Another common shortcut is to use filename *wildcards*. These are a way to specify multiple files at one time. Two common and useful wildcards are the asterisk (`*`) and the question mark (`?`). These wildcards work much like their equivalents in a DOS or Windows command line. Use them when you specify a filename, but not a command—to run a command you have to enter the command name precisely. The asterisk matches any character or set of characters, including none at all. For instance, if you specify `F*D` as a filename, this matches `FD`, `FeD`, `FreeBSD`, or any other filename that begins with *F* and ends with *D*. The question mark, by contrast, matches a single character, so `F?D` matches `FeD` and `FFD`, but not `FD` or `FreeBSD`.

More wildcards are available in FreeBSD than just the asterisk and question mark. One common expansion is to place a set of characters to be matched in square brackets (`[ ]`). If any of the characters in the brackets appear in that position, the wildcard matches. For instance, `F[aeu]D` matches `FaD`, `FeD`, and `FuD`, but not `FoD` or `FeeD`. Instead of listing individual characters, you can specify a range, as in `F[a-c]D`, which matches `FaD`, `FbD`, and `FcD`.

Running Text-Mode Programs

To run a program, you need only type its name at your shell's command prompt. For instance, to run a program called `someprogram`, type **`someprogram`**, and then press ENTER. If the program is in a normal location for programs, the shell loads it and runs it. Depending upon what the program does, it may take over your screen, display some data, ask for input, or create no visible output. Many UNIX programs produce no output when they operate correctly; only when something goes wrong do they produce error messages.

When you type a program's name, shells look in several directories to find the program. These directories constitute the *path*, which is set via an environment variable called `PATH` in your shell's configuration files (including both your personal configuration file and a system-wide configuration file). Typically, the path includes `/bin`, `/usr/bin`, `/usr/X11R6/bin`, `/usr/local/bin`, and possibly some other directories. The `root` account's path usually includes `/sbin` and `/usr/sbin`. These directories hold programs that the system administrator may need to run, but that normal users seldom run. Despite these added directories, `root`'s path is often shorter than that of ordinary users. This serves as an incentive to avoid using the `root` account unnecessarily; without directories for common user tools in `root`'s path, it's awkward to run these programs as `root`.

If a program isn't on the path, you must type the complete path to the program file. For instance, if `someprogram` is in `/opt/bin`, you need to type **`/opt/bin/`**

`someprogram` to run it. Alternatively, you can use a *relative* path, in which the file is located relative to the current directory. Relative paths don't begin with a leading slash (`/`), but take any of several other forms:

- The name may begin with a double dot (`..`), which is the indicator for a given directory's parent. For instance, in `/home/bjolitz`, the `..` refers to `/home`. Thus, from that directory, `../../opt/bin/someprogram` is equivalent to `/opt/bin/someprogram`.
- Another form for a relative path is as a subdirectory of the current directory. For instance, from `/opt`, you might type just **`bin/someprogram`** to launch `/opt/bin/someprogram`.
- A leading `./` indicates that the program file resides in the current directory. Sometimes a current directory (`.`) entry appears in the path, and if this is the case you can omit the `./` characters.

Caution *Using the current directory (`.`) entry in the path is risky because it means that FreeBSD executes any program in the current directory as if it were in a standard location for programs, which are normally writable only by `root`. If you decide to place this entry in the path, do so only at the* end *of the path, so that the shell searches for common programs in their normal locations before looking in the current directory. This advice is especially important for `root` because if `root` has a current directory entry in the path, miscreants could trick `root` into running unauthorized programs stored in the miscreants' home directories. Overall, it's best not to place the current directory entry in any user's path.*

- A leading tilde (~) indicates a user's home directory, so `~/someprogram` refers to `someprogram` in the user's home directory.

These conventions apply to any file specification, not just those used to launch programs. For instance, if you have to type the name of a file upon which a program operates on the same line as the command, you can use any of these methods. In addition, you can omit the leading `./` when referring to a data file in the current directory. You may want to experiment with these techniques using `cd` (to change to a new directory) and `ls` (to display the contents of the current directory or the one you specify). If you get lost using `cd`, type **`cd`** alone to return to your home directory, or **`pwd`** (short for *print working directory*) to find out where you are.

Many programs accept *options*, *parameters*, or *arguments*, which are names for additional information fed to a command on the same line as the command itself. For instance, you might pass the name of a file to an editor to have the editor load the file when it loads. Other parameters are program-specific; they change the way the program operates in some way. The upcoming section, "File Manipulation Commands," includes information on the parameters these commands accept.

If you're new to UNIX-like OSs, you should be aware that FreeBSD is a *case-sensitive* OS, meaning that the case of letters in commands and filenames is important. For instance, typing **pwd** tells you what your current directory is, but typing **PWD** is likely to produce a command not found error message. Most commands use case-sensitive options, as well, although there are a few exceptions to this rule.

File Manipulation Commands

This section describes several commands used to manipulate files. You can use these commands to perform basic actions such as listing files in a directory, moving files, and deleting them. Some other actions are quite important, but are covered elsewhere in this book. In particular, Chapter 5 covers using text editors; and Chapter 8 covers ownership, permissions, and additional miscellaneous file manipulation tools.

cd: Change Current Directory

The cd command moves you into another directory. For instance, if you've got a subdirectory called taxes in your home directory, you can move into this directory by typing **cd taxes** (or **cd ~/taxes** if you're not currently in your home directory). Although it's seldom strictly necessary to change your current directory, doing so can reduce the amount of typing you need to do if you plan to operate on several files in a given directory, by eliminating the need to type extended paths to specify these files.

The cd command is built into whatever shell you use; unlike most commands, it's not a separate program.

ls: List Files

If you want to see the files in a directory, use ls. Its basic output resembles this:

```
$ ls
GNUstep         Mail            XF86Config.new  xinit.core
```

Each filename appears without further information. You have several ways to modify the output of ls to provide additional information. One of these is the -l option, which causes ls to create a *long* listing, thus:

```
$ ls -l
total 406
drwxr-xr-x  5 rodsmith  users     512 Dec 15 22:25 GNUstep
drwx------  2 rodsmith  users     512 Jan  1 20:51 Mail
-rw-r--r--  1 rodsmith  users    3263 Dec 15 21:35 XF86Config.new
-rw-------  1 rodsmith  users  397312 Dec 15 20:50 xinit.core
```

Additional information in this listing includes the permissions on the file (also known as the file's *mode*, signified by the string of ten characters at the start of each line), the username (rodsmith in this example) and group (users) associated with the file, the file's size, and the file's creation date and time (if the file is older than a year, the date format changes to show the year rather than the time).

If you add the name of a directory to the command, it displays information on files in that directory. If you add a partial name with a wildcard, the command displays information on all files or directories that match the wildcard. For instance, typing **`ls *.txt`** displays all the files with names ending in `.txt`.

The `ls` command supports a large number of options. Consult the `ls` man page (by typing **`man ls`**) for information on all of them. The more commonly used options include:

- **`-a`** This option causes `ls` to display all the files in a directory. Normally, `ls` doesn't show so-called *dot files*, which are files whose names begin with dots, such as `.tcshrc`. Dot files are usually configuration files for the programs whose names they otherwise resemble, such as the `tcsh` shell for `.tcshrc`.
- **`-R`** If you want to see all the files in a directory and its subdirectories, use this option, which creates such a recursive listing. Note that this command may create a huge listing, so you may need to pipe the results through a paging tool such as `less`, as in **`ls -R /usr | less`**, to make sense of the output.
- **`-F`** The default short output provides no clues about the type of each file in the listing. Using this option causes FreeBSD to display a character after certain files to indicate their types: a slash (`/`) for directories, an asterisk (`*`) for program files, an at-sign (`@`) for symbolic links, and a few others for more specialized file types.

Note *A symbolic link is a pointer to a file or directory that's stored under another name or in another directory. Symbolic links consume little disk space but enable you to call files or directories by multiple names.*

- **`-f`** Normally, `ls` sorts its entries alphabetically. This option disables this sorting.
- **`-n`** This option is used in conjunction with `-l`; it causes usernames and group names to be replaced by the underlying user IDs (UIDs) and group IDs (GIDs), respectively.

You can combine multiple options in a single option string. You can also add a path (relative or absolute) to a directory whose contents you want to list. For instance, the following command lists all the files and provides file type information for files in the `/tmp` directory:

```
$ ls -aF /tmp
./   .X0-lock    .sawfish-rodsmith/  sample-file.txt@  temprog*
../  .X11-unix/  orbit-rodsmith/     sample.txt
```

cp: Copy Files

The `cp` command copies files. You can use the command in two ways:

```
cp [options] source-file target-file
cp [options] source-file target-directory
```

In the first case, you specify the precise filename to be used for the copy. This name may include a path to the file, or it may be a new filename in your current directory. In the second form, you specify only a directory to which the file will be copied, and `cp` uses the same filename within that directory as the original file used.

Like `ls`, `cp` supports a number of options, enabling it to perform recursive copies (that is, copy an entire directory tree), handle symbolic links, and so on. Chapter 8 covers these advanced `cp` options.

mv: Move Files

The `mv` command moves a file from one location or name to another. Its syntax and use are similar to that of `cp`, and as with `cp`, you can use either a complete filename for the target or a directory to which the file will be moved with its original name intact.

If you're familiar with DOS or Windows text-mode commands, you should be aware that `mv` performs the jobs of two separate DOS commands: `MOVE` and `RENAME`. When you give `mv` a target directory, it works much like the DOS `MOVE` command, moving the file to a new directory. When you provide a target filename, `mv` renames the file to use the new name. If you give a target directory *and* specify a filename within that directory, `mv` does both. For instance, consider the following three commands:

```
$ mv file.txt /tmp
$ mv file.txt /tmp/somefile.txt
$ mv file.txt somefile.txt
```

The first command copies `file.txt` to the `/tmp` directory, leaving its name unchanged. The second command copies and renames the file, and the third renames it within the current directory. When you specify a target directory without a filename, you can move multiple files by using a wildcard or by listing the files individually, thus:

```
$ mv file.txt morefile?.txt /tmp
```

This command copies `file.txt` and any file matching the `morefile?.txt` wildcard to `/tmp`.

When you move a file between directories on a single partition or removable disk device, the file isn't rewritten. Instead, a directory entry for the file is created in the new location and the old directory entry is deleted. Thus, moves within a partition are very fast. If you move a file between partitions, though, FreeBSD must copy the original and then delete it. Thus, these operations tend to be slower, although you'll notice the difference only on very large files.

rm: Remove Files

If a file is no longer needed, you can delete it with the `rm` command. You can also use wildcards to delete many files at once. Chapter 8 covers this command's more advanced options, which enable you to perform tasks such as deleting an entire directory tree, requesting confirmation before each deletion, or overwriting the files' contents before deleting them.

Caution *By default, `rm` does* not *prompt for confirmation before deleting files. This fact can make `rm` a very dangerous tool, particularly in the hands of the superuser. In combination with the option to delete an entire directory tree, `root` can easily remove all the files on a FreeBSD system with a single `rm` command. Although `rm` is a necessary tool, you should think carefully before issuing an `rm` command, particularly as `root`.*

mkdir: Make Directory

FreeBSD treats directories much like files. In fact, directories *are* files, but they're files that contain the names of other files and pointers to them. When you want to create a directory, though, you need a special tool, which is known as `mkdir`. This command's syntax is

```
mkdir [-pv] [-m mode] directory-name...
```

For instance, typing **`mkdir textfiles`** creates a directory called `textfiles` in the current directory. Naturally, you can use a path specification as part of the directory name. The options to this command are:

- **`-p`** Normally, `mkdir` requires that the immediate parent directory of the specified directory exists. For instance, if you type **`mkdir ~/dir1/dir2`** and `~/dir1` doesn't exist, `mkdir` returns an error message. The `-p` option causes `mkdir` to create any necessary parent directories instead, so this command would create `~/dir1` before creating `~/dir1/dir2`.
- **`-v`** This option increases the verbosity of the command's output; it reports every directory that it creates.
- **`-m`** ***`mode`*** Every file and directory has a set of permissions, or *mode*. This option allows you to specify the mode of the directory, as described in Chapter 8.

rmdir: Remove Directory

If you want to remove a directory, you can do so with the `rmdir` command. Type this command followed by the name of the directory you want removed. The directory must be empty, or `rmdir` won't remove it. If you type the `-p` option prior to the directory name, though, `rmdir` will remove the entire directory path you specify. For instance, **`rmdir -p dir1/dir2`** removes `dir2` and then `dir1`, provided both directories are empty when `rmdir` gets around to them.

Tip *If you want to remove an entire directory tree, including directories that aren't empty, you can use the* `-r` *option to* `rm`*, as described in Chapter 8.*

A Quick Tour of the GUI Desktop

You can accomplish a great deal in FreeBSD from a text-mode login. In fact, for some purposes, such as a dedicated server, you might not want or need to run the X Window System (or X for short). Most FreeBSD workstations, though, do run X, so you should become comfortable with it. This section introduces X's GUI environment, including starting X, manipulating files, running programs, and changing the environment's settings.

Note *X is an unusually flexible GUI environment. Depending upon your installation options and personal settings, you may find an environment that's totally unlike the one described here. Chapter 13 covers X configuration options in more detail.*

Starting X

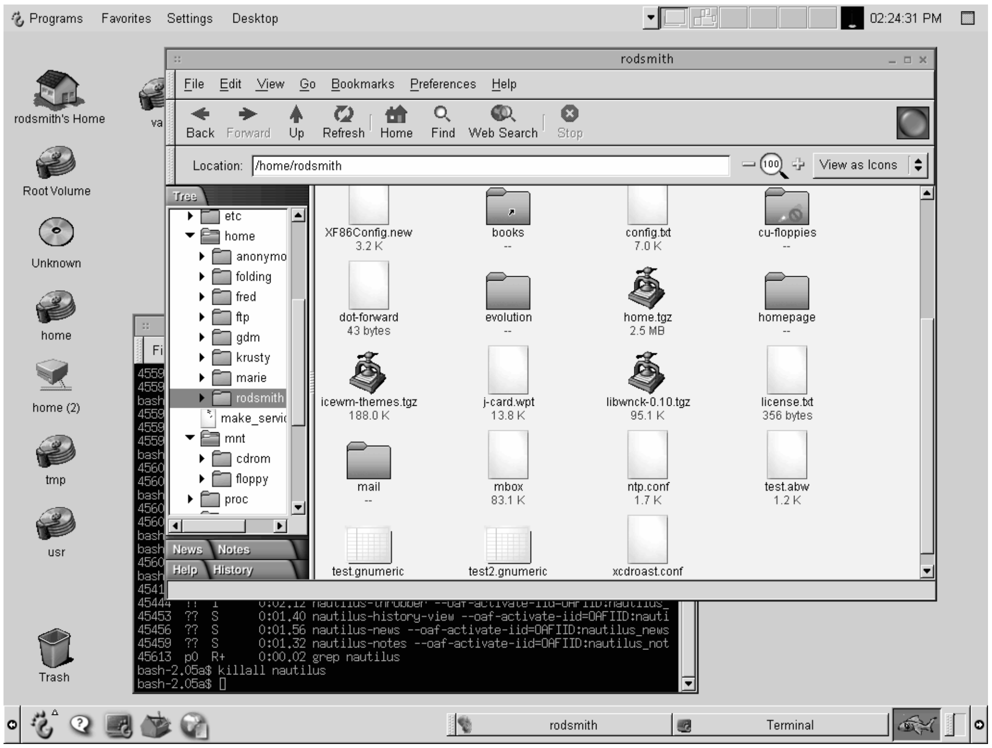
If your FreeBSD system boots up into a GUI login prompt, such as the one shown in Figure 3-1, you needn't do anything explicit to start X; it's already running, and you'll see an X-based interface of some sort when you log in. This mode isn't the default configuration for FreeBSD, though. As described earlier in this chapter, FreeBSD runs in text mode immediately after starting by default. You can change this behavior by running XDM or a similar program, as described in Chapter 21. In the meantime, you can start X from a text-mode login. The usual method of doing this is to type **`startx`**. This action starts X and launches the default GUI environment, which you set when you installed FreeBSD. (You can change this default by editing the `.xinitrc` file, as described in Chapter 13.)

Note *Some users need to add options to* `startx` *to get X running in a desirable resolution or color depth. Chapter 13 covers changing your default X configuration so this shouldn't be necessary.*

After you type **`startx`**, you may see several lines of text scroll across the screen, or the screen may clear or change to another virtual terminal. If all goes well, the screen will then clear and be replaced by a GUI mode screen. Depending upon the environment you installed, you may see some startup messages, or you may simply see a blank screen or one with a few windows or other widgets on it. If you installed the GNOME desktop environment, your display should resemble that shown in Figure 3-3, although your system may not start the windows or show the icons along the left edge of Figure 3-3. The next few pages provide an overview of GNOME, and so may not apply directly if you've chosen to use KDE or a bare window manager. Many features are common, or at least similar, across different desktop environments, but many details also differ.

Note *If X doesn't start correctly, examine the last few lines of output for clues about what went wrong, and consult Chapters 13 and 32.*

You should begin your explorations by testing the features of the window manager and desktop environment. Most FreeBSD window managers work much like the GUI



```
45444  ??  I      0:02.12 nautilus-throbber --oaf-activate-iid=OAFIID:nautilus_
45453  ??  S      0:01.40 nautilus-history-view --oaf-activate-iid=OAFIID:nauti
45456  ??  S      0:01.56 nautilus-news --oaf-activate-iid=OAFIID:nautilus_news
45459  ??  S      0:01.32 nautilus-notes --oaf-activate-iid=OAFIID:nautilus_not
45613  p0  R+     0:00.02 grep nautilus
bash-2.05a$ killall nautilus
bash-2.05a$ 
```

Figure 3-3. *The GNOME environment is similar to that provided by Windows, Mac OS, or other modern GUI OSs.*

in Windows or Mac OS. Here are some important differences between the many FreeBSD window managers and desktop environments:

- **Window focus** A window is said to be *in focus* when it receives input from the keyboard or mouse. Such windows are typically indicated by a different color in their title bars (at the tops of the windows). Usually, the window that's in focus is also the *front* window—the one that's fully exposed. For instance, in Figure 3-3, the file manager window (titled rodsmith) is both in front and in focus. Most window managers shift a window in front and give it focus when you click anywhere in the window, but some shift focus when the mouse moves over the window or only when you click the title bar. Experiment to learn what yours does.
- **Window manager widgets** A *widget* is a GUI tool that lets you interact with a program. Window manager widgets are typically located at the right and left sides of the title bar. Some work much like those in Windows, but others don't.
- **Context menus** Most FreeBSD window managers provide menus with options relating to program launching, exiting from the window manager, and so on. You can often reach these by right-clicking on the desktop. Other environments, including GNOME, provide a menu you can reach by clicking near a corner. In GNOME, the G-shaped foot icon in the lower-left of the screen serves this function.
- **Status bars, menus, docks, and panels** Most desktop environments and some window managers provide some form of menu that's fixed along the top, bottom, or side of the screen. In Figure 3-3, GNOME is configured with menus (which it calls the *Panel*) along both the top and bottom of the screen. In the case of GNOME, menu items at the top enable you to select GNOME options and change between virtual desktops (virtual screens that can hold different programs, reducing clutter, much like virtual terminals in text-mode logins). The bottom panel includes the GNOME Foot, from which you can launch various programs, icons with which you can launch a few particularly important programs, and a button for each program that's running in the current virtual desktop.

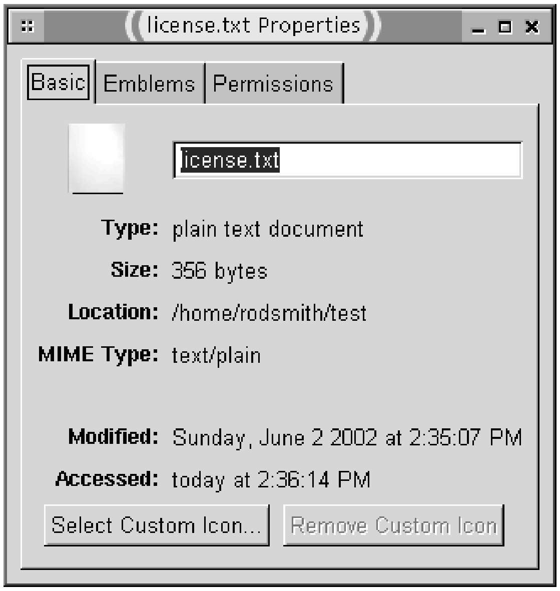
Tip *If your experiments damage your working environment, you can shut down X and delete your user-level configuration files for your environment from your home directory. For GNOME, delete any directory whose name begins with* `.gnome`*, including* `.gnome` *and* `.gnome-desktop`*. If you delete these directories and restart X, you'll find your GNOME desktop restored to its defaults.*

Manipulating Files

If you see a file browser window similar to Figure 3-3's rodsmith window open on your screen, you can use it to move, copy, and otherwise manipulate files. If you don't see it, but do see a *user*'s Home icon on the screen for your home directory (such as Figure 3-3's rodsmith's Home icon), you can open the file browser window by double-clicking that icon. If you don't see the icon and are running GNOME, select GNOME Foot | Programs | Applications | Nautilus to launch it, or type **`nautilus`** in a Terminal window like the one eclipsed by the file manager window in Figure 3-3.

You can move to any directory on the system by clicking Tree in the left-hand side of the window and selecting the desired directory in the resulting directory list. (Figure 3-3 shows this list in its expanded form.) If you want to move to a subdirectory, you can expand a directory by clicking the triangle icon next to the directory's name. Some operations require you to have two windows open. You can do this by selecting File | New Window or by launching a new window using the *user*'s Home icon. Some of the things you can do with GNOME's file manager include the following:

- **Copy files** You can right-click a file and select Duplicate from the resulting pop-up menu to copy it. The copy has the same name as the original, but with `(copy)` inserted into the name. Alternatively, you can right-click an icon and drag it to another directory in the same or another window (including in the directory list to the left of the icons). Doing so produces a menu that enables you to move the file, copy the file, create a symbolic link to the original file, or cancel the operation.
- **Move files** You can move a file by right-clicking it and dragging it, much as with a copy.
- **Delete files** Right-click a file and select Move to Trash to move the file to the Nautilus trash directory. Alternatively, you can drag the file to the trash can icon (near the lower left of the screen in Figure 3-3). These operations move the file to a special trash directory that you can subsequently "empty" by selecting File | Empty Trash from a Nautilus file browser window. You can move the file back out of the trash before you empty it, however, if you find you need the file after all.
- **Rename files** Right-click a file and then select Rename from the context menu to rename the file. You can also rename a file by right-clicking it and selecting Show Properties. The resulting dialog box, shown in Figure 3-4, enables you to rename the file or do other things to it.
- **Change permissions** Right-click the file, select Show Properties, and click the Permissions tab in the Properties dialog box (Figure 3-4) to change the file's permissions. You can use this feature to restrict or loosen access to the file, as described in Chapter 8.
- **Launch programs** If GNOME associates a file type with an application, you can launch that application and have it load the file by double-clicking a file of the appropriate type in the browser window. To launch some other application to load a file, right-click the file and move your mouse over Open With. Nautilus responds by displaying programs it's configured to associate with the file type. (Chapter 23 describes adding associations for particular file types.) Select a specific program to launch that program and have it load the file.



license.txt

Type: plain text document
Size: 356 bytes
Location: /home/rodsmith/test
MIME Type: text/plain
Modified: Sunday, June 2 2002 at 2:35:07 PM
Accessed: today at 2:36:14 PM

Figure 3-4. *The Properties dialog box lets you rename a file, change its group, and modify its permissions, among other things.*

The KDE environment offers a similar file browser, which doubles as a web browser. FreeBSD supports many more file browsers in addition to these two, and you can use the GNOME or KDE browser even if you're not using the rest of the environment.

Running Programs from a GUI

You can launch programs from a GUI environment in several different ways, including

- **Program icons on the panel** The icons immediately to the right of the GNOME Foot in Figure 3-3 provide quick access to four common programs: The GNOME Help Browser, the GNOME Terminal (which gives you access to a text-mode command line), the GNOME Control Center (described in the next section, "Configuring the Environment"), and the Mozilla web browser.
- **Program menu items** Both the GNOME Foot panel item in the lower-left corner of the screen and the Programs menu item in the upper-left corner of the screen give you access to a variety of common programs, divided into several categories.
- **File browser** As noted earlier, you can launch a program from the file browser, either by locating it in your filesystem and double-clicking it or by double-clicking a file that's associated with the program.

- **Desktop icons** Some desktop icons may launch programs. Most of the default icons shown in Figure 3-3 open Nautilus windows on various hard disk partitions, but you can reconfigure the desktop to include icons for programs you like.
- **Terminal launch** You can open a Terminal session using one of the previous methods and then launch a program by typing its name in the Terminal. The Terminal works much like a text-mode login; it runs within it a copy of your default text-mode shell (`sh`, `tcsh`, `bash`, or whatever you've chosen to use). Thus, you can use the Terminal to run text-mode programs.

Note *The GNOME Terminal is one of a class of programs that's used to run a text-based shell and other text-based programs within X. The most generic of these programs is called* `xterm`*, and this class of programs is sometimes referred to by that name. Because GNOME's Terminal is only one of several* `xterm`*-like programs, I use the name xterm to refer to them generically in this book.*

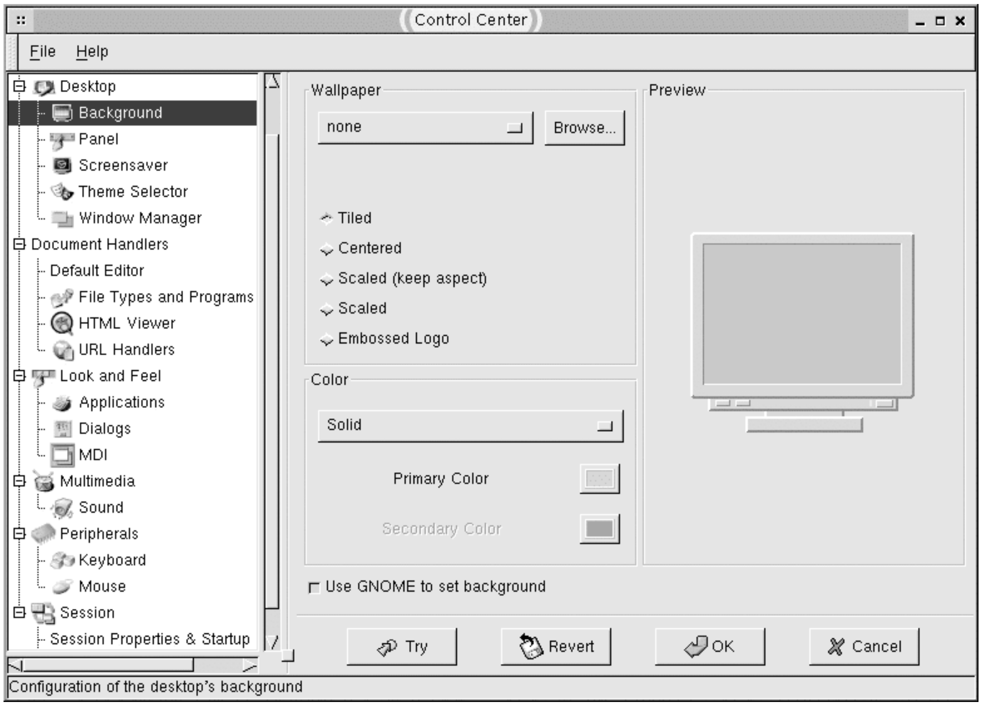
When you run X-based programs, they launch into their own windows, although if you launch such a program from an xterm, the X-based program still "owns" the xterm's text display. Thus, the xterm becomes unusable unless you launch the program with a trailing ampersand (`&`) to shunt it into the background, as described earlier in this chapter, in "Text-Mode and GUI Logins." It's possible for text-based programs to be configured to run from menus or the like, but they're usually run by typing their command names in an xterm window.

Because window environments differ so much, this book emphasizes the xterm approach to launching programs. If you find the program in question on one of your menus, of course, you can launch it in this way. You can even customize your desktop environment to add the program to a menu or as a desktop icon.

One quirk of FreeBSD (or X and the programs that have developed around it, to be more precise) is that GUI programs vary greatly in "look and feel." This is because X is composed of layers, from the X server to the desktop environment. Some of these layers are ones that you choose, such as the window manager, but others are elements that the programmer who develops a program chooses. When two programmers select different development kits, their programs often differ in details such as the fonts used in menus and the appearance of buttons in dialog boxes.

Configuring the Environment

Chances are you'll want to reconfigure some elements of your desktop environment almost immediately. For instance, you might find the keyboard repeat rate to be ridiculous, or you might want to add an interesting background image to the desktop. These details are set in different ways in different GUI environments. In GNOME, you set them in the Control Center, which you can launch by clicking the icon that looks like a toolbox in the panel at the bottom of the screen. If you prefer, you can type **`gnomecc`** in an xterm to launch the Control Center. Figure 3-5 shows the GNOME Control Center with one of its modules loaded.



Configuration of the desktop's background

Figure 3-5. *The GNOME Control Center enables you to adjust GNOME's settings and those of affiliated programs.*

To use the Control Center, you select the general type of setting you want to adjust from the list to the left. Figure 3-5 shows the Desktop | Background module, in which you can set the color of your desktop's background or configure it to use a graphics file instead. As you can see from Figure 3-5, these settings are broken into several broad categories, such as

- **Desktop** This category includes options to set background color or image, adjust the behavior of the Panel, enable a screen saver, load a theme that sets various options in a single operation, and change the window manager that GNOME uses.
- **Document handlers** You can tell GNOME what text editor you prefer to use, what file types to associate with what programs, how it should display HTML (web) documents, and what programs it should use for particular types of Uniform Resource Locators (URLs).

- **Look and feel** The modules in this section enable you to set options relating to the user interface of GNOME programs. These options don't apply to most non-GNOME programs, though, so don't be surprised if they don't affect all your programs.
- **Multimedia** You can configure some sound card settings and associate sounds with particular events, such as logging in, logging out, and receiving e-mail.
- **Peripherals** You can adjust the keyboard repeat rate, mouse tracking speed, mouse handedness, and similar options with these modules.
- **Session** You can set assorted login and logout defaults using the items under the Session option, such as programs you want to launch when you log in and whether GNOME should ask you for confirmation when you log out.

One thing that's important to keep in mind about these configuration options is that they're *user* configuration options. They can vary from one user to another, so several people can use the same computer without having to compromise on options. If Sam really likes a bright pink background, but Sally prefers dark blue, they need not compromise on gray or purple.

Because the GNOME Control Center sets only user options, you can't set some options that affect the entire system. For instance, you can't load drivers, install programs, or format disks from the GNOME Control Center. These actions are administrative tasks that affect the entire system, and they're covered in other chapters of this book.

Logging Off and Shutting Down

When you're done using FreeBSD, you should do one or both of two things: log off and/or shut down. Logging off leaves the computer running, including any servers you may be running, but it closes access to your account; to use your account, you'll have to log in again. Shutting down the computer is more drastic; it terminates all programs and sends the system into a state in which it's safe for you to turn the computer off.

Caution *You should* never *turn the computer off without first shutting down FreeBSD. Like most modern OSs, FreeBSD performs many operations in the background, and* caches *disk accesses—that is, it holds onto data before writing it, in the hope of being able to combine the write into one operation, thus improving performance. These characteristics mean that if you simply shut the computer off, you may lose data, and your filesystems may become corrupted. This can slow the boot process and lose even more data.*

Text-Mode and GUI Methods of Logging Off

Logging off prevents people who might wander by from sitting down at the computer and using your account to do things you might not like, such as send offensive e-mail in your name or break into others' computers. Logging off closes all the programs you're running, including those running in the background.

If you've used a text-mode login, you can log off by typing **`logout`**. This logs you out of the account and displays a new `login:` prompt on the screen. You can log in again at a later time, or somebody else can log in. If you've logged into several virtual terminals, be sure to log out of all of them.

Tip *By default, FreeBSD does* not *clear the screen as part of the logout process. If any sensitive data appear on your screen, you may want to clear the screen by typing* **`clear`** *before you type* **`logout`**.

If you're running X, typing **`logout`** in an xterm window won't work because these windows aren't login sessions. Instead, you must shut down X or log out of your X session. You can accomplish either task by selecting a special logout option on an X menu. Precisely what this option is called varies from one environment to another. In GNOME, it's GNOME Foot | Log Out. In some stand-alone window managers, you can find the logout option in a menu you obtain by right-clicking on the desktop. Sometimes this option is called *Close Window Manager*, *Exit X*, or something similar.

After you select the X-based logout option, you'll either see the XDM login screen (if your system is configured to use XDM) or your original text mode screen, with quite a few messages displayed by the X server. (You can safely ignore these, if your X session worked properly.) You can then type **`logout`** to log off the computer.

Note *In some cases, X doesn't terminate correctly after you select the desktop environment or window manager logout option. In such cases, you can press* CTRL-ALT-BACKSPACE *to terminate X.*

Text-Mode and GUI Methods of Shutting Down

Shutting down the computer allows you to safely power it off. You have to be logged in as `root` to shut down the computer (an ordinary user login followed by `su` to acquire `root` privileges also works). The command involved is called `shutdown`. You can issue it from a text-mode login or from an xterm inside X. Its syntax is

```
shutdown [options] time [warning-message]
```

The *`time`* is the time when the system shuts down, and it can take one of three forms:

- **`now`** The string `now` represents an immediate shutdown. You might use this time specification to shut down a workstation that's being used by a single person.
- ***`+number`*** To schedule a shutdown some number of minutes in the future, specify the number of minutes by preceding it with a plus sign. For instance, `+30` indicates a shutdown in half an hour.

- ***yymmddhhmm*** To schedule a shutdown for a particular time, specify it as a ten-digit number, starting with a two-digit year and moving down through the month, day, hour, and minute. (You must use a 24-hour format for the hour.)

The optional *`warning-message`* is a text message that appears on all users' text-mode login consoles starting at ten hours before the scheduled shutdown time. For instance, you might pass **`"Shut down for disk upgrade; up at 7:00 AM"`** as the warning message, to let users know why the system is being shut down and when they might expect it to be up again.

Finally, but appearing immediately after the `shutdown` command when you type it, the program accepts any of several options. These are:

- **`-h`** The system is halted after the shutdown. This form of shutdown enables you to power off or reboot the computer manually.
- **`-p`** The system is halted and powered off after the shutdown. This option requires that your hardware support software-controlled power off, as do most modern computers. (Some older systems don't support this.)
- **`-r`** The system is halted and rebooted after the shutdown. You might use this option to reboot into another OS, or after making changes to the FreeBSD kernel.
- **`-k`** This option kicks all users off the system and disallows further logins, but doesn't actually shut down the system. You might do this if you want to make extensive software changes or perform tests in a controlled environment.
- **`-o`** The shutdown procedure with `-h`, `-p`, or `-r` normally involves the `shutdown` process calling `init`, which does the actual work. This option causes `shutdown` to do the work itself. It's normally not required.
- **`-n`** This option can be used in conjunction with `-o`, and causes `shutdown` to skip flushing the filesystem cache. This method can cause disk corruption, so it's not something you'd normally want to use.
- **- (Dash)** If you include a single dash as an option, `shutdown` prompts you for a warning message rather than using one you provide on the same command line.

After you issue a `shutdown` command, FreeBSD reverses many of the steps it took during startup. FreeBSD terminates any running programs, including X, any servers that are running, and so on. The system unmounts network filesystems and partitions it's mounted, and then performs whatever shutdown action you specified. If you used `-h`, you'll see a message stating that the system has been halted; at that point, you can power it off or reboot it by pressing the Reset switch on the computer's case. If you used `-p` or `-r`, the system automatically powers down (if your hardware supports this option) or reboots, respectively.

An alternative to the `shutdown` command is to press CTRL-ALT-DELETE at the console's keyboard. This action shuts down and reboots the FreeBSD system, and you may find it easier to remember this keystroke than the details of the `shutdown` command. This alternative may therefore be a good one if the system is a workstation used by inexperienced users. Of course, if the intent is to shut down rather than reboot the computer, you'll have to hit the power switch before FreeBSD begins booting, or you'll risk file corruption. Some replacements for the XDM GUI login program enable users to shut down the computer from the console. These programs provide a shutdown option from a button or menu item. Both the GDM and KDM programs (parts of the GNOME and KDE environments, respectively) provide this option.

When to Shut Down

If you participate in Usenet newsgroup or mailing list discussions, sooner or later you'll come across a perennial debate: When is it appropriate to shut down a computer? The two schools on this issue are the never-shut-it-down school and the shut-down-to-save-power school.

The argument against shutting down a computer is that component failures are most likely to occur because of the stresses involved in turning devices on and off. The changes in heat and voltage put more wear and tear on a device than does constant operation. Consider light bulbs. They usually fail when you apply power, not when they've been turned on for a while. The same is true of computer components, so the lifetime of a computer may be extended by leaving it on all the time.

An opposing view holds that the extended life you might gain from leaving a computer on at all times is minor, particularly when you consider that the components in question are likely to become obsolete long before they fail. Leaving the computer on continuously consumes power, though, and the power savings is more important (by this argument) than the extension in component life.

Both of these arguments have merit, and you'll have to decide between them for yourself. One further factor, though, lies in the FreeBSD automatic maintenance tools. FreeBSD runs certain programs late at night to handle some routine maintenance issues, such as cleaning old files out of the `/tmp` directory, as described in Chapter 28. Thus, if you decide to routinely shut down your computer at night, you should leave it running overnight every once in a while.

Of course, all of this assumes that you're running a workstation. Most servers need to run 24 hours a day, 7 days a week, and so are shut down only for hardware upgrades, repairs, or the like.

Summary

FreeBSD is distinguishable from Windows soon after you press the power switch on the computer. The FreeBSD startup sequence produces a great deal of textual output, summarizing the steps it's taking while booting your system. After this, the default FreeBSD configuration presents a text-mode login screen, and logging in lets you run text-mode programs. FreeBSD's UNIX heritage makes it important that you understand at least some of these commands, such as those to view and manipulate files. To run GUI programs, you must start X by typing a single command (**`startx`**), which loads X and whatever X environment you've chosen as the default or configured for yourself as an individual user. Some FreeBSD GUI environments are very similar to those of Windows or other GUI OSs, so if you're familiar with such tools, you shouldn't have too much trouble adjusting to FreeBSD. When you're done, you should log off of your account to reduce the risk of it being abused by a passer-by. When you need to turn the computer off or reboot it, you should be sure to use a proper shutdown procedure, rather than simply hitting the power switch or Reset button.