preparing to do in any case. Although not involved in the litigation, it was suggested to FreeBSD that they should also move to 4.4BSD-Lite, which was done with the release of FreeBSD release 2.0 in late 1994.

Now, in the early 21st century, FreeBSD is the best known of the BSD operating systems, one that many consider to follow in the tradition of the CSRG. I can think of no greater honour for the development team. It was developed on a shoestring budget, yet it manages to outperform commercial operating systems by an order of magnitude.

The end of the UNIX wars

In the course of the FreeBSD project, a number of things have changed about UNIX. Sun Microsystems moved from a BSD base to a System V base in the late 80s, a move that convinced many people that BSD was dead and that System V was the future. Things turned out differently: in 1992, AT&T sold USL to Novell, Inc., who had introduced a product based on System V.4 called UnixWare. Although UnixWare has much better specifications than SCO's old System V.3 UNIX, it was never a success, and Novell finally sold their UNIX operation to SCO. SCO itself was then bought out by Caldera (which recently changed its name back to SCO), while the ownership of the UNIX trade mark has passed to the Open Group. System V UNIX is essentially dead: current commercial versions of UNIX have evolved so far since System V that they can't be considered the same system. By contrast, BSD is alive and healthy, and lives on in FreeBSD, NetBSD, OpenBSD and Apple's Mac OS X.

The importance of the AT&T code in the earlier versions of FreeBSD was certainly overemphasized in the lawsuit. All of the disputed code was over 10 years old at the time, and none of it was of great importance. In January 2002, Caldera released all ''ancient'' versions of UNIX under a BSD license. These specifically included all versions of UNIX from which BSD was derived: the first to seventh editions of Research UNIX and 32V, the predecessor to 3BSD. As a result, all versions of BSD, including those over which the lawsuit was conducted, are now freely available.

Other free UNIX-like operating systems

FreeBSD isn't the only free UNIX-like operating system available—it's not even the best-known one. The best-known free UNIX-like operating system is undoubtedly Linux, but there are also a number of other BSD-derived operating systems. We'll look at them first:

- *386/BSD* was the original free BSD operating system, introduced by William F. Jolitz in 1992. It never progressed beyond a test stage: instead, two derivative operating systems arose, FreeBSD and NetBSD. 386/BSD has been obsolete for years.
- *NetBSD* is an operating system which, to the casual observer, is almost identical to FreeBSD. The main differences are that NetBSD concentrates on hardware independence, whereas FreeBSD concentrates on performance. FreeBSD also tries harder to be easy to understand for a beginner. You can find more information about NetBSD at *http://www.NetBSD.org*.

- *OpenBSD* is a spin-off of NetBSD that focuses on security. It's also very similar to FreeBSD. You can find more information at *http://www.OpenBSD.org*.
- Apple computer introduced Version 10 (X) of its *Mac OS* in early 2001. It is a big deviation from previous versions of Mac OS: it is based on a Mach microkernel with a BSD environment. The base system (Darwin) is also free. FreeBSD and Darwin are compatible at the user source code level.

You could get the impression that there are lots of different, incompatible BSD versions. In fact, from a user viewpoint they're all very similar to each other, much more than the individual distributions of Linux, which we'll look at next.

FreeBSD and Linux

In 1991, Linus Torvalds, then a student in Helsinki, Finland, decided he wanted to run UNIX on his home computer. At that time the BSD sources were not freely available, and so Linus wrote his own version of UNIX, which he called Linux.

Linux is a superb example of how a few dedicated, clever people can produce an operating system that is better than well-known commercial systems developed by a large number of trained software engineers. It is better even than a number of commercial UNIX systems.

Obviously, I prefer FreeBSD over Linux, or I wouldn't be writing this book, but the differences between FreeBSD and Linux are more a matter of philosophy rather than of concept. Here are a few contrasts:

Table 1-1: Differences between FreeBSD and Linux

FreeBSD	Linux
FreeBSD is a direct descendent of the original UNIX, though it contains no residual AT&T code.	Linux is a clone and never contained any AT&T code.
FreeBSD is a complete operating system, maintained by a central group of software developers under the Concurrent Versions System which maintains a complete history of the project development. There is only one distribution of FreeBSD.	Linux is a kernel, personally maintained by Linus Torvalds and a few trusted companions. The non-kernel programs supplied with Linux are part of a *distribution*, of which there are several. Distributions are not completely compatible with each other.
The FreeBSD development style emphasizes accountability and documentation of changes.	The Linux kernel is maintained by a small number of people who keep track of all changes. Unofficial patches abound.
The kernel supplied with a specific release of FreeBSD is clearly defined.	Linux distributions often have subtly different kernels. The differences are not always documented.

If you're running X, you can use a browser like *mozilla* to read the documents. If you don't have X running yet, use *lynx*. Both of these programs are included in the CD-ROM distribution. To install them, use *sysinstall*, which is described on page 92.

lynx is not a complete substitute for complete web browsers such as *mozilla*: since it is text-only, it is not capable of displaying the large majority of web pages correctly. It's good enough for reading most of the FreeBSD online documentation, however.

In each case, you start the browser with the name of the document, for example:

```
$ lynx /usr/share/doc/en/books/handbook/index.html
$ mozilla /usr/share/doc/en/books/handbook/index.html &
```

Enter the & after the invocation of *mozilla* to free up the window in which you invoke it: *mozilla* opens its own window.

If you haven't installed the documentation, you can still access it from the Live Filesystem CD-ROM. Assuming the CD-ROM is mounted on */cdrom*, choose the file */cdrom/usr/share/doc/en/books/handbook/index.html*.

Alternatively, you can print out the handbook. This is a little more difficult, and of course you'll lose the hypertext references, but you may prefer it in this form. To format the handbook for printing, you'll need a PostScript printer or *ghostscript*. See page 271 for more details of how to print PostScript.

The printable version of the documentation doesn't usually come with the CD-ROM distribution. You can pick it up with *ftp* (see page 433) from *ftp://ftp.FreeBSD.ORG/pub/FreeBSD/doc/*, which has the same directory structure as described above. For example, you would download the handbook in PostScript form from *ftp://ftp.FreeBSD.ORG/pub/FreeBSD/doc/en/books/handbook/book.ps.bz2*.

The online manual

The most comprehensive documentation on FreeBSD is the online manual, usually referred to as the *man pages*. Nearly every program, file, library function, device or interface on the system comes with a short reference manual explaining the basic operation and various arguments. If you were to print it out, it would run to well over 8,000 pages.

When online, you view the man pages with the command *man*. For example, to learn more about the command *ls*, type:

```
$ man ls
LS(1)                    FreeBSD Reference Manual                    LS(1)

NAME
     ls - list directory contents

SYNOPSIS
     ls [-ACFLRTacdfiloqrstu1] [ file ... ]

DESCRIPTION
     For each operand that names a file of a type other than directory, ls
```

```
     displays its name as well as any requested, associated information.  For
     each operand that names a file of type directory, ls displays the names.
(etc)
```

In this particular example, with the exception of the first line, the text in **`constant width bold`** is not input, it's the way it appears on the screen.

The online manual is divided up into sections numbered:

1. User commands
2. System calls and error numbers
3. Functions in the C libraries
4. Device drivers
5. File formats
6. Games and other diversions
7. Miscellaneous information
8. System maintenance and operation commands
9. Kernel interface documentation

In some cases, the same topic may appear in more than one section of the online manual. For example, there is a user command *chmod* and a system call `chmod()`. In this case, you can tell the *man* command which you want by specifying the section number:

```
$ man 1 chmod
```

This command displays the manual page for the user command chmod. References to a particular section of the online manual are traditionally placed in parentheses in written documentation. For example, *chmod(1)* refers to the user command *chmod*, and *chmod(2)* means the system call.

This is fine if you know the name of the command and forgot how to use it, but what if you can't recall the command name? You can use *man* to search for keywords in the command descriptions by using the `-k` option, or by starting the program *apropos*:

```
$ man -k mail
$ apropos mail
```

Both of these commands do the same thing: they show the names of the man pages that have the keyword *mail* in their descriptions.

Alternatively, you may browse through the */usr/bin* directory, which contains most of the system executables. You'll see lots of file names, but you don't have any idea what they do. To find out, enter one of the lines:

```
$ cd /usr/bin; man -f *
$ cd /usr/bin; whatis *
```

Both of these commands do the same thing: they print out a one-line summary of the purpose of the program:

```
$ cd /usr/bin; man -f *
a2p(1)           - Awk to Perl translator
addftinfo(1)     - add information to troff font files for use with groff
apply(1)         - apply a command to a set of arguments
apropos(1)       - search the whatis database
...etc
```

Printing man pages

If you prefer to have man pages in print, rather than on the screen, you can do this in two different ways:

- The simpler way is to redirect the output to the spooler:

  ```
  $ man ls | lpr
  ```

  This gives you a printed version that looks pretty much like the original on the screen, except that you may not get bold or underlined text.

- You can get typeset output with *troff*:

  ```
  $ man -t ls | lpr
  ```

  This gives you a properly typeset version of the man page, but it requires that your spooling system understand PostScript—see page 271 for more details of printing PostScript, even on printers that don't understand PostScript.

GNU info

The Free Software Foundation has its own online hypertext browser called *info*. Many FSF programs come with either no man page at all, or with an excuse for a man page (*gcc*, for example). To read the online documentation, you need to browse the *info* files with the *info* program, or from *Emacs* with the *info* mode. To start *info*, simply type:

```
$ info
```

In *Emacs*, enter **CTRL-h i** or **ALT-x** `info`. Whichever way you start *info*, you can get brief introduction by typing **h**, and a quick command reference by typing **?**.

Other documentation on FreeBSD

FreeBSD users have access to probably more top-quality documentation than just about any other operating system. Remember that word UNIX is trademarked. Sure, the lawyers tell us that we can't refer to FreeBSD as UNIX, because UNIX belongs to the Open Group. That doesn't make the slightest difference to the fact that nearly every book on UNIX applies more directly to FreeBSD than any other flavour of UNIX. Why?

Commercial UNIX vendors have a problem, and FreeBSD doesn't help them: why should people buy their products when you can get it free from the FreeBSD Project (or, for that matter, from other free UNIX-like operating systems such as NetBSD, OpenBSD and Linux)? One obvious reason would be ''value-added features.'' So they add features or fix weak points in the system, put a copyright on the changes, and help lock their customers in to their particular implementation. As long as the changes are really useful, this is legitimate, but it does make the operating system less compatible with ''standard UNIX,'' and the books about standard UNIX are less applicable.

In addition, many books are written by people with an academic background. In the UNIX world, this means that they are more likely than the average user to have been exposed to BSD. Many general UNIX books handle primarily BSD, possibly with an additional chapter on the commercial System V version.

In Appendix A, *Bibliography*, you'll find a list of books that I find worthwhile. I'd like to single out some that I find particularly good, and that I frequently use myself:

- *UNIX Power Tools*, by Jerry Peek, Tim O'Reilly, and Mike Loukides, is a superb collection of interesting information, including a CD-ROM. Recommended for everybody, from beginners to experts.
- *UNIX for the Impatient*, by Paul W. Abrahams and Bruce R. Larson, is more similar to this book, but it includes a lot more material on specific products, such as shells and the *Emacs* editor.
- The *UNIX System Administration Handbook*, by Evi Nemeth, Garth Snyder, Scott Seebass, and Trent R. Hein, is one of the best books on systems administration I have seen. It covers a number different UNIX systems, including an older version of FreeBSD.

There are also many active Internet groups that deal with FreeBSD. Read about them in the online handbook.

The FreeBSD community

FreeBSD was developed by a world-wide group of developers. It could not have happened without the Internet. Many of the key players have never even met each other in person; the main means of communication is via the Internet. If you have any kind of Internet connection, you can participate as well. If you don't have an Internet connection, it's about time you got one. The connection doesn't have to be complete: if you can receive email, you can participate. On the other hand, FreeBSD includes all the software you need for a complete Internet connection, not the very limited subset that most PC-based ''Internet'' packages offer you.

Mailing lists

As it says in the copyright, FreeBSD is supplied as-is, without any support liability. If you're on the Internet, you're not alone, however. Liability is one thing, but there are plenty of people prepared to help you, most for free, some for fee. A good place to start is with the mailing lists. There are a number of mailing lists that you can join. Some of the more interesting ones are:

- `FreeBSD-questions@FreeBSD.org` is the list to which you may send general questions, in particular on how to use FreeBSD. If you have difficulty understanding anything in this book, for example, this is the right place to ask. It's also the list to use if you're not sure which is the most appropriate.
- `FreeBSD-newbies@FreeBSD.org` is a list for newcomers to FreeBSD. It's intended for people who feel a little daunted by the system and need a bit of reassurance. It's not the right place to ask any kind of technical question.
- `FreeBSD-hackers@FreeBSD.org` is a technical discussion list.
- `FreeBSD-current@FreeBSD.org` is an obligatory list for people who run the development version of FreeBSD, called `FreeBSD-CURRENT`.
- `FreeBSD-stable@FreeBSD.org` is a similar list for people who run the more recent stable version of FreeBSD, called `FreeBSD-STABLE`. We'll talk about these versions on page 582. Unlike the case for `FreeBSD-CURRENT` users, it's not obligatory for `FreeBSD-STABLE` users to subscribe to `FreeBSD-stable`.

You can find a complete list of FreeBSD mailing lists on the web site, currently at *http://www.FreeBSD.org/doc/en_US.ISO8859-1/books/handbook/eresources.html*. This address is part of the online handbook and may change when the handbook is modified; follow the link *Mailing Lists* from *http://www.FreeBSD.org/* if it is no longer valid, or if you can't be bothered typing in the URI.

The mailing lists are run by *mailman* (in the Ports Collection). Join them via the web interface mentioned above. You will receive a mail message from *mailman* asking you to confirm your subscription by replying to the message. You don't need to put anything in

the reply: the reply address is used once only, and you're the only person who will ever see it, so the system knows that it's you by the fact that you replied at all. You also have the option of confirming via a web interface with a specially generated URI. Similar considerations apply in this case.

FreeBSD mailing lists can have a very high volume of traffic. The FreeBSD-questions mailing list, for example, has thousands of subscribers, and many of them are themselves mailing lists. It receives over a hundred messages every day. That's about a million messages a day in total for just one mailing list, so when you sign up for a mailing list, be sure to read the charter. You can find the URI from the *mailman* confirmation message. It's also a good idea to ''lurk'' (listen, but not say anything) on the mailing list a while before posting anything: each list has its own traditions.

When submitting a question to `FreeBSD-questions`, consider the following points:

1. Remember that nobody gets paid for answering a FreeBSD question. They do it of their own free will. You can influence this free will positively by submitting a well-formulated question supplying as much relevant information as possible. You can influence this free will negatively by submitting an incomplete, illegible, or rude question. It's perfectly possible to send a message to FreeBSD-questions and not get an answer even if you follow these rules. It's much more possible to not get an answer if you don't.

2. Not everybody who answers FreeBSD questions reads every message: they look at the subject line and decide whether it interests them. Clearly, it's in your interest to specify a subject. ''FreeBSD problem'' or ''Help'' aren't enough. If you provide no subject at all, many people won't bother reading it. If your subject isn't specific enough, the people who can answer it may not read it.

3. When sending a new message, well, send a new message. Don't just reply to some other message, erase the old content and change the subject line. That leaves an `In-Reply-To:` header which many mail readers use to thread messages, so your message shows up as a reply to some other message. People often delete messages a whole thread at a time, so apart from irritating people, you also run a chance of having the message deleted unread.

4. Format your message so that it is legible, and PLEASE DON'T SHOUT!!!!!. It's really painful to try to read a message written full of typos or without any line breaks. A lot of badly formatted messages come from bad mailers or badly configured mailers. The following mailers are known to send out badly formatted messages without you finding out about them:

   Eudora
   exmh
   Microsoft Exchange
   Microsoft Internet Mail
   Microsoft Outlook
   Netscape

As you can see, the mailers in the Microsoft world are frequent offenders. If at all possible, use a UNIX mailer. If you must use a mailer under Microsoft environments, make sure it is set up correctly. Try not to use MIME: a lot of people use mailers which don't get on very well with MIME.

For further information on this subject, check out *http://www.lemis.com/email.html*.

5. Make sure your time and time zone are set correctly. This may seem a little silly, since your message still gets there, but many of the people you are trying to reach get several hundred messages a day. They frequently sort the incoming messages by subject and by date, and if your message doesn't come before the first answer, they may assume they missed it and not bother to look.

6. Don't include unrelated questions in the same message. Firstly, a long message tends to scare people off, and secondly, it's more difficult to get all the people who can answer all the questions to read the message.

7. Specify as much information as possible. This is a difficult area: the information you need to submit depends on the problem. Here's a start:

   - If you get error messages, don't say ''I get error messages'', say (for example) ''I get the error message *No route to host*''.
   - If your system panics, don't say ''My system panicked'', say (for example) ''my system panicked with the message *free vnode isn't*''.
   - If you have difficulty installing FreeBSD, please tell us what hardware you have, particularly if you have something unusual.
   - If, for example, you have difficulty getting PPP to run, describe the configuration. Which version of PPP do you use? What kind of authentication do you have? Do you have a static or dynamic IP address? What kind of messages do you get in the log file? See Chapter 20, *Configuring PPP*, for more details in this particular case.

8. If you don't get an answer immediately, or if you don't even see your own message appear on the list immediately, don't resend the message. Wait at least 24 hours. The FreeBSD mailer offloads messages to a number of subordinate mailers around the world. Usually the messages come through in a matter of seconds, but sometimes it can take several hours for the mail to get through.

9. If you do all this, and you still don't get an answer, there could be other reasons. For example, the problem is so complicated that nobody knows the answer, or the person who does know the answer was offline. If you don't get an answer after, say, a week, it might help to re-send the message. If you don't get an answer to your second message, though, you're probably not going to get one from this forum. Resending the same message again and again will only make you unpopular.

How to follow up to a question

Often you will want to send in additional information to a question you have already sent. The best way to do this is to reply to your original message. This has three advantages:

1. You include the original message text, so people will know what you're talking about. Don't forget to trim unnecessary text, though.
2. The text in the subject line stays the same (you did remember to put one in, didn't you?). Many mailers will sort messages by subject. This helps group messages together.
3. The message reference numbers in the header will refer to the previous message. Some mailers, such as mutt, can thread messages, showing the exact relationships between the messages.

There are more suggestions, in particular for answering questions, at *http://www.lemis.com/questions.html*. See also Chapter 26, *Electronic mail: clients* for more information about sending mail messages. You may also like to check out the FreeBSD web site at *http://www.FreeBSD.org/* and the support page at *http://www.FreeBSD.org/support.html*.

In addition, a number of companies offer support for FreeBSD. See the web page *http://www.FreeBSD.org/commercial/consulting_bycat.html* for some possibilities.

Unsubscribing from the mailing lists

There's a lot of traffic on the mailing lists, particularly on `FreeBSD-questions`. You may find you can't take it and want to get out again. Again, you unsubscribe from the list either via the web or via a special mail address, ***not*** by sending mail to the the list. Each message you get from the mailing lists finishes with the following text:

```
freebsd-questions@freebsd.org mailing list
http://lists.freebsd.org/mailman/listinfo/freebsd-questions
To unsubscribe, send any mail to "freebsd-questions-unsubscribe@freebsd.org"
```

Don't be one of those people who send the unsubscribe request to the mailing list instead.

User groups

But how about meeting FreeBSD users face to face? There are a number of user groups around the world. If you live in a big city, chances are that there's one near you. Check *http://www.FreeBSD.org/support.html#user* for a list. If you don't find one, consider taking the initiative and starting one.

In addition, USENIX holds an annual conference, the *BSDCon*, which deals with technical aspects of the BSD operating systems. It's also a great opportunity to get to know other users from around the world. If you're in Europe, there is also a BSDCon Europe, which at the time of writing was not run by USENIX. See *http://www.eurobsdcon.org* for more details.

Reporting bugs

If you find something wrong with FreeBSD, we want to know about it, so that we can fix it. To report a bug, use the *send-pr* program to send it as a mail message.

There used to be a web form at *http://www.FreeBSD.org/send-pr.html*, but it has been closed down due to abuse.

The Berkeley daemon

The little daemon at the right symbolizes BSD. It is included with kind permission of Marshall Kirk McKusick, one of the leading members of the former Computer Sciences Research Group at the University of California at Berkeley, and owner of the daemon's copyright. Kirk also wrote the foreword to this book.

The daemon has occasionally given rise to a certain amount of confusion. In fact, it's a joking reference to processes that run in the background—see Chapter 8, *Taking control*, page 150, for a description. The outside world occasionally sees things differently, as the following story indicates:

```
Newsgroups: alt.humor.best-of-usenet
Subject: [comp.org.usenix] A Great Daemon Story

From: Rob Kolstad <kolstad@bsdi.com>
Newsgroups: comp.org.usenix
Subject: A Great Daemon Story
```

Linda Branagan is an expert on daemons. She has a T-shirt that sports the daemon in tennis shoes that appears on the cover of the 4.3BSD manuals and *The Design and Implementation of the 4.3BSD UNIX Operating System* by S. Leffler, M. McKusick, M. Karels, J. Quarterman, Addison Wesley Publishing Company, Reading, MA 1989.

She tells the following story about wearing the 4.3BSD daemon T-shirt:

Last week I walked into a local ‘‘home style cookin' restaurant/watering hole'' in Texas to pick up a take-out order. I spoke briefly to the waitress behind the counter, who told me my order would be done in a few minutes.

So, while I was busy gazing at the farm implements hanging on the walls, I was approached by two ‘‘natives.'' These guys might just be the original Texas rednecks.

‘‘Pardon us, ma'am. Mind if we ask you a question?''

Well, people keep telling me that Texans are real friendly, so I nodded.

‘‘Are you a Satanist?''

Well, at least they didn't ask me if I liked to party.

‘‘Uh, no, I can't say that I am.''

‘‘Gee, ma’am. Are you sure about that?’’ they asked.

I put on my biggest, brightest Dallas Cowboys cheerleader smile and said, ‘‘No, I’m positive. The closest I’ve ever come to Satanism is watching Geraldo.’’

‘‘Hmmm. Interesting. See, we was just wondering why it is you have the lord of darkness on your chest there.’’

I was this close to slapping one of them and causing a scene—then I stopped and noticed the shirt I happened to be wearing that day. Sure enough, it had a picture of a small, devilish-looking creature that has for some time now been associated with a certain operating system. In this particular representation, the creature was wearing sneakers.

They continued: ‘‘See, ma’am, we don’t exactly appreciate it when people show off pictures of the devil. Especially when he’s lookin’ so friendly.’’

These idiots sounded terrifyingly serious.

Me: ‘‘Oh, well, see, this isn’t really the devil, it’s just, well, it’s sort of a mascot.

Native: ‘‘And what kind of football team has the devil as a mascot?’’

Me: ‘‘Oh, it’s not a team. It’s an operating—uh, a kind of computer.’’

I figured that an ATM machine was about as much technology as these guys could handle, and I knew that if I so much as uttered the word ‘‘UNIX’’ I would only make things worse.

Native: ‘‘Where does this satanical computer come from?’’

Me: ‘‘California. And there’s nothing satanical about it really.’’

Somewhere along the line here, the waitress noticed my predicament—but these guys probably outweighed her by 600 pounds, so all she did was look at me sympathetically and run off into the kitchen.

Native: ‘‘Ma’am, I think you’re lying. And we’d appreciate it if you’d leave the premises now.’’

Fortunately, the waitress returned that very instant with my order, and they agreed that it would be okay for me to actually pay for my food before I left. While I was at the cash register, they amused themselves by talking to each other.

Native #1: ‘‘Do you think the police know about these devil computers?’’

Native #2: ‘‘If they come from California, then the FBI oughta know about ’em.’’

They escorted me to the door. I tried one last time: ‘‘You’re really blowing this all out of proportion. A lot of people use this ‘kind of computers.’ Universities, researchers, businesses. They’re actually very useful.’’

Big, big, *big* mistake. I should have guessed at what came next.

Native: ‘‘Does the government use these devil computers?’’

Me: ‘‘Yes.’’

Another *big* boo-boo.

Native: ‘‘And does the government pay for ’em? With our tax dollars?’’

I decided that it was time to jump ship.

Me: ‘‘No. Nope. Not at all. Your tax dollars never entered the picture at all. I promise. No sir, not a penny. Our good Christian congressmen would never let something like that happen. Nope. Never. Bye.’’

Texas. What a country.

The daemon tradition goes back quite a way. As recently as 1996, after the publication of the first edition of this book, the following message went through the `FreeBSD-chat` mailing list:

To: "Jonathan M. Bresler" <jmb@freefall.freebsd.org>
Cc: obrien@antares.aero.org (Mike O’Brien),
joerg_wunsch@uriah.heep.sax.de,
chat@FreeBSD.org, juphoff@tarsier.cv.nrao.edu
Date: Tue, 07 May 1996 16:27:20 -0700
Sender: owner-chat@FreeBSD.org

> details and gifs PLEASE!

If you insist. `:-)`

Sherman, set the Wayback Machine for around 1976 or so (see Peter Salus’ *A Quarter Century of UNIX* for details), when the first really national UNIX meeting was held in Urbana, Illinois. This would be after the ‘‘forty people in a Brooklyn classroom’’ meeting held by Mel Ferentz (yeah I was at that too) and the more-or-less simultaneous West Coast meeting(s) hosted by SRI, but before the UNIX Users Group was really incorporated as a going concern.

I knew Ken Thompson and Dennis Ritchie would be there. I was living in Chicago at the time, and so was comic artist Phil Foglio, whose star was just beginning to rise. At that time I was a bonded locksmith. Phil’s roommate had unexpectedly split town, and he was the only one who knew the combination to the wall safe in their apartment. This is the only apartment I’ve ever seen that had a wall safe, but it sure did have one, and Phil had some stuff locked in there. I didn’t hold out much hope, since safes are far beyond where I was (and am) in my locksmithing sphere of competence, but I figured ‘‘no guts no glory’’ and told him I’d give it a whack. In return, I told him, he could do some T-shirt art for me. He readily agreed.

Wonder of wonders, this safe was vulnerable to the same algorithm that Master locks used to be susceptible to. I opened it in about 15 minutes of manipulation. It was my greatest moment as a locksmith and Phil was overjoyed. I went down to my lab and shot some Polaroid snaps of the PDP-11 system I was running UNIX on at the time, and gave it to Phil with some descriptions of the visual puns I wanted: pipes, demons with forks running along the pipes, a ‘‘bit bucket’’ named */dev/null*, all that.

What Phil came up with is the artwork that graced the first decade’s worth of ‘‘UNIX T-shirts,’’ which were made by a Ma and Pa operation in a Chicago suburb. They turned out transfer art using a 3M color copier in their basement. Hence, the PDP-11 is reversed (the tape drives are backwards) but since Phil left off the front panel, this was hard to tell. His trademark signature was photo-reversed, but was

recopied by the T-shirt people and ‘‘re-forwardized,’’ which is why it looks a little funny compared to his real signature.

Dozens and dozens of these shirts were produced. Bell Labs alone accounted for an order of something like 200 for a big picnic. However, only four (4) REAL originals were produced: these have a distinctive red collar and sleeve cuff. One went to Ken, one to Dennis, one to me, and one to my then-wife. I now possess the latter two shirts. Ken and Dennis were presented with their shirts at the Urbana conference.

People ordered these shirts direct from the Chicago couple. Many years later, when I was living in LA, I got a call from Armando Stettner, then at DEC, asking about that now-famous artwork. I told him I hadn’t talked to the Illinois T-shirt makers in years. At his request I called them up. They’d folded the operation years ago and were within days of discarding all the old artwork. I requested its return, and duly received it back in the mail. It looked strange, seeing it again in its original form, a mirror image of the shirts with which I and everyone else were now familiar.

I sent the artwork to Armando, who wanted to give it to the Ultrix marketing people. They came out with the Ultrix poster that showed a nice shiny Ultrix machine contrasted with the chewing-gum-and-string PDP-11 UNIX people were familiar with. They still have the artwork, so far as I know.

I no longer recall the exact contents of the letter I sent along with the artwork. I did say that as far as I knew, Phil had no residual rights to the art, since it was a ‘work made for hire’, though nothing was in writing (and note this was decades before the new copyright law). I do not now recall if I explicitly assigned all rights to DEC. What is certain is that John Lassiter’s daemon, whether knowingly borrowed from the original, or created by parallel evolution, postdates the first horde of UNIX daemons by at least a decade and probably more. And if Lassiter’s daemon looks a lot like a Phil Foglio creation, there’s a reason.

I have never scanned in Phil’s artwork; I’ve hardly ever scanned in anything, so I have no GIFs to show. But I have some very very old UNIX T-shirts in startlingly good condition. Better condition than I am at any rate: I no longer fit into either of them.

Mike O’Brien
creaky antique

Note the date of this message: it appeared since the first edition of this book. Since then, the daemon image has been scanned in, and you can find a version at *http://www.mckusick.com/beastie/shirts/usenix.html*.

Incorrect boot installation

It's possible to forget to install the bootstrap, or even to wipe it the existing bootstrap. That sounds like a big problem, but in fact it's easy enough to recover from. Refer to the description of the boot process on page 529, and boot from floppy disk or CD-ROM. Interrupt the boot process with the space bar. You might see:

```
BTX loader 1.00  BTX version is 1.01
BIOS drive A: is disk0
BIOS drive C: is disk1
BIOS drive D: is disk1
BIOS 639kB/130048kB available memory

FreeBSD/i386 bootstrap loader, Revision 0.8
(grog@freebie.example.com, Thu Jun 13 13:06:03 CST 2002)
Loading /boot/defaults/loader.conf

Hit [Enter] to boot immediately, or any other key for command prompt.
Booting [kernel] in 6 seconds...        press space bar here
ok unload                               unload the current kernel
ok set currdev disk1s1a                 and set the location of the new one
ok load /boot/kernel/kernel             load the kernel
ok boot                                 then start it
```

This boots from the drive */dev/ad0s1a*, assuming that you are using IDE drives. The correspondence between the name */dev/ad0s1a* and *disk1s1a* goes via the information at the top of the example: BTX only knows the BIOS names, so you'd normally be looking for the first partition on drive *C:*. After booting, install the correct bootstrap with `bsdlabel -B` or *boot0cfg*, and you should be able to boot from hard disk again.

Geometry problems

Things might continue a bit further: you elect to install *booteasy*, and when you boot, you get the Boot Manager prompt, but it just prints `F?` at the boot menu and won't accept any input. In this case, you may have set the hard disk geometry incorrectly in the partition editor when you installed FreeBSD. Go back into the partition editor and specify the correct geometry for your hard disk. You may need to reinstall FreeBSD from the beginning if this happens.

It used to be relatively common that *sysinstall* couldn't calculate the correct geometry for a disk, and that as a result you could install a system, but it wouldn't boot. Since those days, *sysinstall* has become a lot smarter, but it's still barely possible that you'll run into this problem.

If you can't figure out the correct geometry for your machine, and even if you don't want to run Microsoft on your machine, try installing a small Microsoft partition at the beginning of the disk and install FreeBSD after that. The install program sees the Microsoft partition and tries to infer the correct geometry from it, which usually works. After the partition editor has accepted the geometry, you can remove the Microsoft partition again. If you are sharing your machine with Microsoft, make sure that the Microsoft partition is before the FreeBSD partition.

Alternatively, if you don't want to share your disk with any other operating system, select the option to use the entire disk (a in the partition editor). You're less likely to have problems with this option.

System hangs during boot

A number of problems may lead to the system hanging during the boot process. All the known problems have been eliminated, but there's always the chance that something new will crop up. In general, the problems are related to hardware probes, and the most important indication is the point at which the boot failed. It's worth repeating the boot with the verbose flag: again, refer to the description of the boot process on page 529. Interrupt the boot process with the space bar and enter:

```
Hit [Enter] to boot immediately, or any other key for command prompt.
Booting [kernel] in 6 seconds...        press space bar here
ok set boot_verbose                     set a verbose boot
ok boot                                 then continue
```

This flag gives you additional information that might help diagnose the problem. See Chapter 29 for more details of what the output means.

If you're using ISA cards, you may need to reconfigure the card to match the kernel, or change the file */boot/device.hints* to match the card settings. See the example on page 608. Older versions of FreeBSD used to have a program called *UserConfig* to perform this function, but it is no longer supported.

System boots, but doesn't run correctly

If you get the system installed to the point where you can start it, but it doesn't run quite the way you want, ***don't reinstall***. In most cases, reinstallation won't help. Instead, try to find the cause of the problem—with the aid of the FreeBSD-questions mailing list if necessary—and fix the problem.

Root file system fills up

You might find that the installation completes successfully, and you get your system up and running, but almost before you know it, the root file system fills up. This is relatively unlikely if you follow my recommendation to have one file system for */*, */usr* and */var*, but if you follow the default recommendations, it's a possibility. It could be, of course, that you just haven't made it big enough—FreeBSD root file systems have got bigger over the years. In the first edition of this book I recommended 32 MB ''to be on the safe side.'' Nowadays the default is 128 MB.

On the other hand, maybe you already have an 128 MB root file system, and it still fills up. In this case, check where you have put your */tmp* and */var* file systems. There's a good chance that they're on the root file system, and that's why it's filling up.

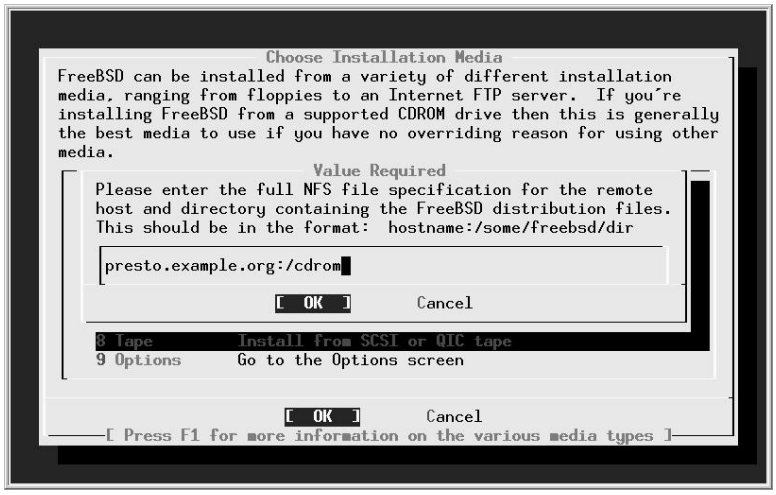


Figure 5-17: Specifying NFS file system

The only required directory is *base*. You can include as many other directories as you want, but be sure to maintain the directory structure. In other words, if you also wanted to install *XF86336* and *manpages*, you would copy them to *C:\FREEBSD\XF86336* and *C:\FREEBSD\MANPAGES*.

Creating floppies for a floppy installation

Installation from floppy disk is definitely the worst choice you have. You will need nearly 50 floppies for the minimum installation, and about 250 for the complete installation. The chance of one of them being bad is high. Most problems on a floppy install can be traced to bad media, or differences in alignment between the media and the drive in which they are used, so:

> ***Before starting, format all floppies in the drive you intend to use, even if they are preformatted.***

The first two floppies you'll need are the Kernel floppy and the MFS Root floppy, which were described earlier.

In addition, you need at minimum as many floppies as it takes to hold all files in the *base* directory, which contains the binary distribution. Read the file *LAYOUT.TXT* paying special attention to the ''Distribution format'' section, which describes which files you need.

If you're creating the floppies on a FreeBSD machine, you can put *ufs* file systems on the floppies instead:

```
# fdformat -f 1440 fd0.1440
# bsdlabel -w fd0.1440 floppy3
# newfs -t 2 -u 18 -l 1 -i 65536 /dev/fd0
```

Next, copy the files to the floppies. The distribution files are split into chunks that will fit exactly on a conventional 1.44MB floppy. Copy one file to each floppy. Make very sure to put the file *base.inf* on the first floppy; it is needed to find out how many floppies to read.

The installation itself is straightforward enough: follow the instructions starting on page 63, select *Floppy* in the installation medium menu on page 76, then follow the prompts.

- *Emacs* is the GNU Emacs editor recommended in this book. We'll look at it on page 139. Other popular editors are *vi* (in the base system) and *vim* (in the Ports Collection).
- *fetchmail* is a program for fetching mail from POP mailboxes. We look at it on page 504.
- *fvwm2* is a window manager that you may prefer to a full-blown desktop. We look at it on page 118.
- *galeon* is a web browser. We'll look at it briefly on page 418.
- *ghostscript* is a PostScript interpreter. It can be used to display PostScript on an X display, or to print it out on a non-PostScript printer. We'll look at it on page 273.
- *gpg* is an encryption program.
- *gv* is a utility that works with *ghostscript* to display PostScript on an X display. It allows magnification and paging, both of which *ghostscript* does not do easily. We'll look at it on page 273.
- *ispell* is a spell check program.
- *kde* is the desktop environment recommended in this book. We'll look at it in more detail in Chapter 7, *The tools of the trade*.
- *mkisofs* is a program to create CD-R images. We look at it in chapter Chapter 13, *Writing CD-Rs*.
- *mutt* is the mail user agent (MUA, or mail reader) recommended in Chapter 26, *Electronic mail: clients*.
- *postfix* is the mail transfer agent (MTA) recommended in chapter Chapter 27, *Electronic mail: servers*.
- *xtset* is a utility to set the title of an *xterm* window. It is used by the *.bashrc* file installed with the *instant-workstation* package.
- *xv* is a program to display images, in particular *jpeg* and *gif*.

Why do I recommend these particular ports? Simple: because I like them, and I use them myself. That doesn't mean they're the only choice, though. Others prefer the *Gnome* window manager to *kde*, or the *pine* or *elm* MUAs to *mutt*, or the *vim* editor to *Emacs*. This is the stuff of holy wars. See *http://www.tuxedo.org/~esr/jargon/html/entry/holy-wars.html* for more details.

Instant workstation

The ports mentioned in the previous section are included in the *misc/instant-workstation* port, which installs typical software and configurations for a workstation and allows you to be productive right away. At a later point you may find that you prefer other software, in which case you can install it.

It's possible that the CD set you get will not include *instant-workstation*. That's not such a problem: you just install the individual ports from this list. You can also do this if you don't like the list of ports.

Changing the default shell for root

After installation, you may want to change the default shell for existing users to *bash*. If you have installed *instant-workstation*, you should copy the file */usr/share/skel/dot.bashrc* to `root`'s home directory and call it *.bashrc* and *.bash_profile*. First, start

```
presto# cp /usr/share/skel/dot.bashrc .bashrc
presto# ln .bashrc .bash_profile
presto# bash
=== root@presto (/dev/ttyp2) ~ 1 -> chsh
```

The last command starts an editor with the following content:

```
#Changing user database information for root.
Login: root
Password:
Uid [#]: 0
Gid [# or name]: 0
Change [month day year]:
Expire [month day year]:
Class:
Home directory: /root
Shell: /bin/csh
Full Name: Charlie &
Office Location:
Office Phone:
Home Phone:
Other information:
```

Change the `Shell` line to:

```
Shell: /usr/local/bin/bash
```

Note that the *bash* shell is in the directory */usr/local/bin*; this is because it is not part of the base system. The standard shells are in the directory */bin*.

Adding users

A freshly installed FreeBSD system has a number of users, nearly all for system components. The only login user is `root`, and you shouldn't log in as `root`. Instead you should add at least one account for yourself. If you're transferring a *master.passwd* file from another system, you don't need to do anything now. Otherwise select this item and then the menu item `User`, and fill out the resulting menu like this:

This change of disk ID can be a problem. One of the first things you do with a new disk is to create new disk labels and file systems. Both offer excellent opportunities to shoot yourself in the foot if you choose the wrong disk: the result would almost certainly be the complete loss of data on that disk. Even apart from such catastrophes, you'll have to edit */etc/fstab* before you can mount any file systems that are on the disk. The alternatives are to wire down the device names, or to change the SCSI IDs. In FreeBSD 5.0, you wire down device names and busses by adding entries to the boot configuration file */boot/device.hints*. We'll look at that on page 574.

Formatting the disk

Formatting is the process of rewriting every sector on the disk with a specific data pattern, one that the electronics find most difficult to reproduce: if they can read this pattern, they can read anything. Microsoft calls this a *low-level format*.[1] Obviously it destroys any existing data, so

> ***If you have anything you want to keep, back it up before formatting.***

Most modern disks don't need formatting unless they're damaged. In particular, formatting will not help if you're having configuration problems, if you can't get PPP to work, or you're running out of disk space. Well, it *will* solve the disk space problem, but not in the manner you probably desire.

If you do need to format a SCSI disk, use *camcontrol*. *camcontrol* is a control program for SCSI devices, and it includes a lot of useful functions that you can read about in the man page. To format a disk, use the following syntax:

```
# camcontrol format da1
```

> ***Remember that formatting a disk destroys all data on the disk. Before using the command, make sure that you need to do so: there are relatively few cases that call for formatting a disk. About the only reasons are if you want to change the physical sector size of the disk, or if you are getting "medium format corrupted" errors from the disk in response to read and write requests.***

FreeBSD can format only floppies and SCSI disks. In general it is no longer possible to reformat ATA (IDE) disks, though some manufacturers have programs that can recover from some data problems. In most cases, though, it's sufficient to write zeros to the entire disk:

```
# dd if=/dev/zero of=/dev/ad1 bs=128k
```

If this doesn't work, you may find formatting programs on the manufacturer's web site. You'll probably need to run them under a Microsoft platform.

1. Microsoft also uses the term *high-level format* for what we call creating a file system.

Using sysinstall

If you can, use *sysinstall* to partition your disk. Looking at the *dmesg* output for our new disk, we see:

```
da1 at sym1 bus 0 target 0 lun 0
da1: <SEAGATE ST15230W SUN4.2G 0738> Fixed Direct Access SCSI-2 device
da1: 20.000MB/s transfers (10.000MHz, offset 15, 16bit), Tagged Queueing Enabled
da1: 4095MB (8386733 512 byte sectors: 255H 63S/T 522C)
```

You see the standard installation screen (see Chapter 5, page 60). Select `Index`, then `Partition`, and you see the following screen:

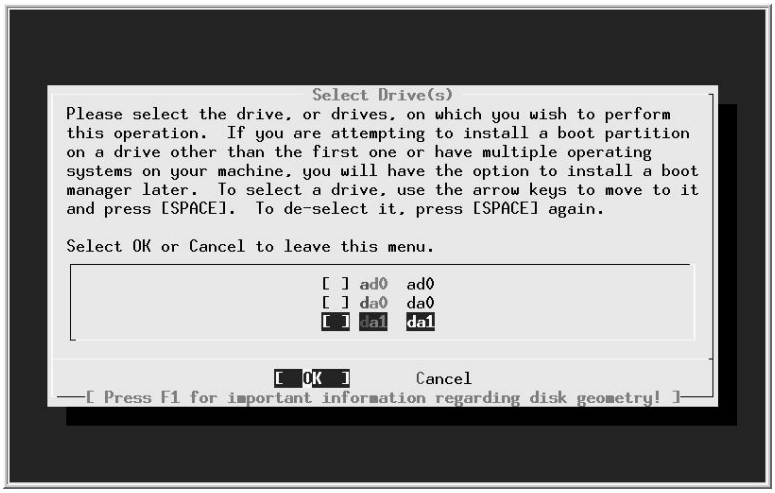


Figure 11-1: Disk selection menu

In this case, we want to partition */dev/da1*, so we position the cursor on `da1` (as shown) and press **Enter**. We see the disk partition menu, which shows that the disk currently contains three partitions:

- The first starts at offset 0, and has a length of 63. This is *not* unused, no matter what the description says. It's the partition table, padded to the length of a ''track.''
- The next partition takes up the bulk of the drive and is a Microsoft partition.
- Finally, we have 803 sectors left over as a result of the partitioning scheme. Sometimes this can be much larger—I have seen values as high as 35 MB. This is the price we pay for compatibility with PC BIOS partitioning.

We want a FreeBSD partition, not a Microsoft partition. At this point, we have a number of choices:

```
    start 2091453, size 6295133 (3073 Meg), flag 0
        beg: cyl 351/ head 0/ sector 1;
        end: cyl 413/ head 12/ sector 47
Are we happy with this entry? [n] y
The data for partition 3 is:
<UNUSED>
Do you want to change it? [n] Enter pressed
The data for partition 4 is:
sysid 165,(FreeBSD/NetBSD/386BSD)
    start 47, size 8386539 (4094 Meg), flag 80 (active)
        beg: cyl 0/ head 1/ sector 1;
        end: cyl 413/ head 12/ sector 47
Do you want to change it? [n] y

The static data for the DOS partition 4 has been reinitialized to:
sysid 165,(FreeBSD/NetBSD/386BSD)
    start 47, size 8386539 (4094 Meg), flag 80 (active)
        beg: cyl 0/ head 1/ sector 1;
        end: cyl 413/ head 12/ sector 47
Supply a decimal value for "sysid (165=FreeBSD)" [165] 0
Supply a decimal value for "start" [47] 0
Supply a decimal value for "size" [8386539] 0
Explicitly specify beg/end address ? [n] Enter pressed
<UNUSED>
Are we happy with this entry? [n] y
Do you want to change the active partition? [n] y
Supply a decimal value for "active partition" [1] 2
Are you happy with this choice [n] y

We haven't changed the partition table yet.  This is your last chance.
parameters extracted from in-core disklabel are:
cylinders=13726 heads=13 sectors/track=47 (611 blks/cyl)

Figures below won't work with BIOS for partitions not in cyl 1
parameters to be used for BIOS calculations are:
cylinders=13726 heads=13 sectors/track=47 (611 blks/cyl)

Information from DOS bootblock is:
1: sysid 6,(Primary 'big' DOS (> 32MB))
    start 0, size 2091453 (1021 Meg), flag 0
        beg: cyl 0/ head 0/ sector 1;
        end: cyl 350/ head 12/ sector 47
2: sysid 165,(FreeBSD/NetBSD/386BSD)
    start 2091453, size 6295133 (3073 Meg), flag 80 (active)
        beg: cyl 351/ head 0/ sector 1;
        end: cyl 413/ head 12/ sector 47
3: <UNUSED>
4: <UNUSED>
Should we write new partition table? [n] y
```

You'll notice a couple of things here:

- Even though we created valid partitions 1 and 2, which cover the entire drive, *fdisk* gave us the phantom partition 4 which covered the whole disk, and we had to remove it.
- The cylinder numbers in the summary at the end don't make any sense. We've already calculated that the Microsoft partition goes from cylinder 0 to cylinder 3422 inclusive, and the FreeBSD partition goes from cylinder 3423 to cylinder 13725. But *fdisk* says that the Microsoft partition goes from cylinder 0 to cylinder 350 inclusive, and the FreeBSD partition goes from cylinder 351 to cylinder 413. What's that all about?

The problem here is overflow: once upon a time, the maximum cylinder value was 1023, and *fdisk* still thinks this is the case. The numbers we're seeing here are the remainder left by dividing the real cylinder numbers by 1024.

Labelling the disk

Once we have a valid PC BIOS partition table, we need to create the file systems. We won't look at the Microsoft partition in any more detail, but we still need to do some more work on our FreeBSD slice (slice or PC BIOS partition 2). It'll make life easier here to remember a couple of things:

- From now on, we're just looking at the slice, which we can think of as a logical disk. Names like *disk label* really refer to the slice, but many standard terms use the word *disk*, so we'll continue to use them.
- All offsets are relative to the beginning of the slice, not the beginning of the disk. Sizes also refer to the slice and not the disk.

The first thing we need is the disk (slice) label, which supplies general information about the slice:

- The fact that it's a FreeBSD slice.
- The size of the slice.
- The sizes, types and layout of the file systems.
- Some obsolete information about details like rotational speed of the disk and the track-to-track switching time. This is still here for historical reasons only. It may go away soon.

The only information we need to input is the kind, size and locations of the partitions. In this case, we have decided to create a file system on partition h (*/dev/da1s2h*) and swap space on partition b (*/dev/da1s1b*). The swap space will be 512 MB, and the file system will take up the rest of the slice. This is mainly tradition: traditionally data disks use the h partition and not the a partition, so we'll stick to that tradition, though there's nothing to stop you from using the a partition if you prefer. In addition, we need to define the c partition, which represents the whole slice. In summary, the FreeBSD slice we want to create looks like:

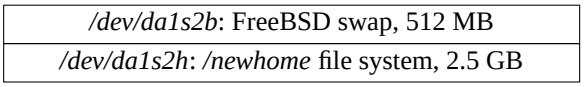


Figure 11-4: FreeBSD slice on second disk

bsdlabel

The program that writes the disk label used to be called *disklabel*. As FreeBSD migrated to multiple platforms, this proved to be too generic: many hardware platforms have their own disk label formats. For example, FreeBSD on SPARC64 uses the Sun standard labels. On platforms which use the old BSD labels, such as the PC, the name was changed to *bsdlabel*. On SPARC64 it is called *sunlabel*. On each platform, the appropriate file is linked to the name *disklabel*, but some of the options have changed. In addition, the output format now normally ignores a number of historical relics. It's not as warty as *fdisk*, but it can still give you a run for your money. You can usually ignore most of the complexity, though. You can normally create a disk label with the single command:

```
# bsdlabel -w /dev/da1s2 auto
```

This creates the label with a single partition, `c`. You can look at the label with *bsdlabel* without options:

```
# bsdlabel /dev/da1s2
# /dev/da0s2:
8 partitions:
#        size   offset    fstype   [fsize bsize bps/cpg]
  c:  6295133        0    unused        0     0         # "raw" part, don't edit
```

At this point, the only partition you have is the ''whole disk'' partition `c`. You still need to create partitions `b` and `h` and specify their location and size. Do this with *bsdlabel -e*, which starts an editor with the output you see above. Simply add additional partitions:

```
8 partitions:
#        size   offset    fstype   [fsize bsize bps/cpg]
  c:  6295133        0    unused        0     0         # "raw" part, don't edit
  b:  1048576        0    swap          0     0
  h:  5246557  1048576    unused        0     0
```

You don't need to maintain any particular order, and you don't need to specify that partition `h` will be a file system. In the next step, *newfs* does that for you automatically.

Problems running bsdlabel

Using the old *disklabel* program used to be like walking through a minefield. Things have got a lot better, but it's possible that some problems are still hiding. Here are some of the problems that have been encountered in the past, along with some suggestions about what to do if you experience them:

- When writing a label (the `-w` option), you may find:

  ```
  # bsdlabel -w da1s2
  bsdlabel: /dev/da1s2c: Undefined error: 0
  ```

  This message may be the result of the kernel having out-of-date information about the slice in memory. If this is the case, a reboot may help.

- `No disk label on disk` is straightforward enough. You tried to use *bsdlabel* to look at the label before you had a label to look at.
- `Label magic number or checksum is wrong!` tells you that *bsdlabel* thinks it has a label, but it's invalid. This could be the result of an incorrect previous attempt to label the disk. It can be difficult to get rid of an incorrect label. The best thing to do is to repartition the disk with the label in a different position, and then copy */dev/zero* to where the label used to be:

  ```
  # dd if=/dev/zero of=/dev/da1 bs=128k count=1
  ```

  Then you can repartition again the way you want to have it.
- `Open partition would move or shrink` probably means that you have specified incorrect values in your slice definitions. Check particularly that the `c` partition corresponds with the definition in the partition table.
- `write: Read-only file system` means that you are trying to do something invalid with a valid disk label. FreeBSD write protects the disk label, which is why you get this message.
- In addition, you might get kernel messages like:

  ```
  fixlabel: raw partition size > slice size
  or
  fixlabel: raw partitions offset != slice offset
  ```

  The meanings of these messages should be obvious.

Creating file systems

Once we have a valid label, we need to create the file systems. In this case, there's only one file system, on */dev/da1s2h*. Mercifully, this is easier:

```
# newfs -U /dev/da1s2h
/dev/vinum/da1s2h: 2561.8MB (5246556 sectors) block size 16384, fragment size 2048
        using 14 cylinder groups of 183.77MB, 11761 blks, 23552 inodes.
        with soft updates
super-block backups (for fsck -b #) at:
 160, 376512, 752864, 1129216, 1505568, 1881920, 2258272, 2634624, 3010976, 3387328,
 3763680, 4140032, 4516384, 4892736
```

The `-U` flag tells *newfs* to enable soft updates, which we looked at on page 191.

Mounting the file systems

Finally the job is done. Well, almost. You still need to mount the file system, and to tell the system that it has more swap. But that's not much of a problem:

```
# mkdir /newhome                                  make sure we have a directory to mount on
# mount /dev/da1s2h /newhome                      and mount it
# swapon /dev/da1s2b
# df                                              show free capacity and mounted file systems
Filesystem    1024-blocks     Used    Avail Capacity  Mounted on
/dev/ad0s1a         19966    17426      944    95%    /
/dev/ad0s1e       1162062   955758   113340    89%    /usr
procfs                  4        4        0   100%    /proc
presto:/            15823     6734     8297    45%    /presto/root
presto:/usr        912271   824927    41730    95%    /presto/usr
presto:/home      1905583  1193721   521303    70%    /presto/home
presto:/S         4065286  3339635   563039    86%    /S
/dev/da1s2h       2540316        2  2337090     0%    /newhome
# pstat -s                                        show swap usage
Device          1K-blocks     Used    Avail Capacity  Type
/dev/ad0s4b       524160        0   524160     0%    Interleaved
/dev/da1s2b       524160        0   524160     0%    Interleaved
Total            1048320        0  1048320     0%
```

This looks fine, but when you reboot the system, */newhome* and the additional swap will be gone. To ensure that they get mounted after booting, you need to add the following lines to */etc/fstab*:

```
/dev/da1s2b             none            swap    sw      0       0
/dev/da1s2h             /newhome        ufs     rw      0       0
```

Moving file systems

Very frequently, you add a new disk to a system because existing disks have run out of space. Let's consider the disk we have just added and assume that currently the files in */home* are physically located on the */usr* file system, and that */home* is a symbolic link to */usr/home*. We want to move them to the new file system and then rename it to */home*. Here's what to do:

- Copy the files:

  ```
  # cd /home
  # tar cf - . | (cd /newhome; tar xvf - 2>/var/tmp/tarerrors)
  ```

  This writes any error messages to the file */var/tmp/tarerrors*. If you don't do this, any errors will get lost.

- ***Check /var/tmp/tarerrors and make sure that the files really made it to the right place!***

- Remove the old files:

```
# rm -rf /usr/home
```

- In this case, */home* was a symbolic link, so we need to remove it and create a directory called */home*:

```
# rm /home
# mkdir /home
```

  You don't need to do this if */home* was already a directory (for example, if you're moving a complete file system).

- Modify */etc/fstab* to contain a line like:

```
/dev/da1s2h             /home           ufs     rw      0       0
```

- Unmount the */newhome* directory and mount it as */home*:

```
# umount /newhome
# mount /home
```

Recovering from disk data errors

Modern hard disks are a miracle in evolution. Today you can buy a 200 GB hard disk for under $200, and it will fit in your shirt pocket. Thirty years ago, a typical disk drive was the size of a washing machine and stored 20 MB. You would need 10,000 of them to store 200 GB.

At the same time, reliability has gone up, but disks are still relatively unreliable devices. You can achieve maximum reliability by keeping them cool, but sooner or later you are going to run into some kind of problem. One kind is due to surface irregularities: the disk can't read a specific part of the surface.

Modern disks make provisions for recovering from such errors by allocating an alternate sector for the data. IDE drives do this automatically, but with SCSI drives you have the option of enabling or disabling reallocation. Usually reallocation is enabled when you buy the disk, but occasionally it is not. When installing a new disk, you should check that the parameters *ARRE* (*Auto Read Reallocation Enable*) and *AWRE* (*Auto Write Reallocation Enable*) are turned on. For example, to check and set the values for disk */dev/da1*, you would enter:

```
# camcontrol modepage da1 -m 1 -e
```

This command will start up your favourite editor (either the one specified in the `EDITOR` environment variable, or *vi* by default) with the following data:

```
AWRE (Auto Write Reallocation Enbld):  0
ARRE (Auto Read Reallocation Enbld):  0
TB (Transfer Block):  1
EER (Enable Early Recovery):  0
PER (Post Error):  1
DTE (Disable Transfer on Error):  0
DCR (Disable Correction):  0
Read Retry Count:  41
Write Retry Count:  24
```

The values for AWRE and ARRE should both be 1. If they aren't, as in this case, where AWRE is 0, change the data with the editor, write it back, and exit. *camcontrol* writes the data back to the disk and enables the option.

Note the last two lines in this example. They give the number of actual retries that this drive has performed. You can reset these values too if you want; they will be updated if the drive performs any additional retries.

Normal disk installations lay out overlapping disk partitions: the *c* partition overlaps all the other partitions. You can do the same thing with a Vinum drive, which is also a partition. You can then create subdisks in the Vinum drive corresponding in length and position to the partitions. In the following diagram, each column represents the entire disk. On the left there are four normal partitions. In the middle is the *c* partition, and on the right is a Vinum drive partition:

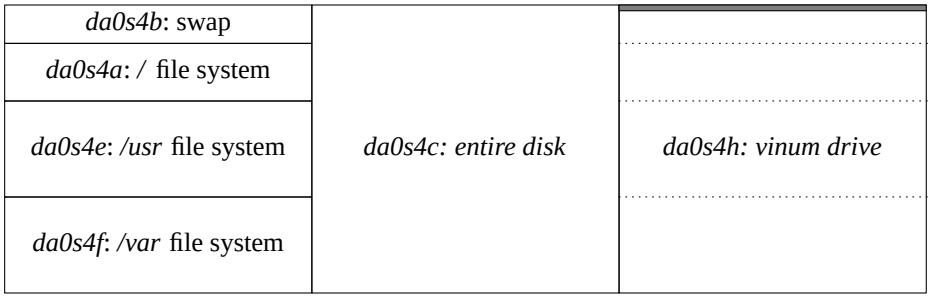


Figure 12-9: Partition layout with Vinum

This layout shows three file system partitions and a swap partition, which is not the layout recommended on page 68. We'll look at the reasons for this below.

The shaded area at the top of the Vinum partition represents the configuration information, which cuts into the swap partition and the bootstrap. To fix that, we redefine the swap partition to start after the Vinum configuration information and to be a total of 281 sectors shorter, 265 sectors for the Vinum configuration and 16 sectors for the bootstrap.

The swap partition isn't normally the first partition on a drive, but you can create this layout with *sysinstall* simply by creating the swap partition before any other partition. Consider installing FreeBSD on a 4 GB drive. Create, in sequence, a swap partition of 256 MB, a root file system of 256 MB, a */usr* file system of 2 GB, and a */var* file system to take up the rest. It's important to create the swap partition at the beginning of the disk, so you create that first. After installation, the output of *bsdlabel* looks like this:

```
a:   524288   524288    4.2BSD     2048 16384 32776               root file system
b:   524288        0      swap                                    swap partition
c:  8385867        0    unused        0     0         # "raw" part, don't edit
d:  4194304  1048576    4.2BSD     2048 16384 28512               /usr file system
e:  3142987  5242880    4.2BSD     2048 16384 28512               /var file system
```

This corresponds to the left and centre columns in the figure above. To convert to Vinum, you need to:

- create a volume of type *vinum* that starts after the bootstrap in the *c* partition.
- shorten the swap partition by 281 sectors at the beginning.

Boot in single user mode and remount the root file system (to make it read/write), mount the */usr* directory and run *bsdlabel* with the `-e` (edit label) option:

```
# mount -u /
# mount /usr
# bsdlabel -e da0s4
```

See page 215 for more information about *bsdlabel*. You need to boot in single user mode because otherwise the swap partition would be mounted, and you can't change the size of the swap partition when it's mounted.

When you finish, the partition table should look like this (changed values in **bold**):

```
#        size   offset    fstype   [fsize bsize bps/cpg]
  a:   524288   524288    4.2BSD     2048 16384 32776           root file system
  b:   524007      281      swap                                 swap partition
  c:  8385867        0    unused        0     0         # "raw" part, don't edit
  d:  4194304  1048576    4.2BSD     2048 16384 28512           /usr file system
  e:  3142987  5242880    4.2BSD     2048 16384 28512           /var file system
  h:  8385851       16     vinum                                 Vinum drive
```

Be sure to shorten the length of the swap partition by 281 sectors, or it will overlap the root partition and cause extreme data corruption when swap gets full, which might not happen until months later.

The next step is to create the Vinum objects. The plexes and volumes are straightforward: each plex is concatenated with a single subdisk, and the volume has a single plex. It makes sense to give the volumes names that relate to the mount point.

Creating the subdisks requires a little more care. You can use the size values from the *bsdlabel* output above directly in a Vinum configuration file, but since the Vinum drive starts after the bootstrap, you need to subtract 16 (the length of the bootstrap) from the offset values:

```
drive rootdev device /dev/da0s4h
volume root
  plex org concat
# a:       524288               524288    4.2BSD     2048 16384 32776
    sd len 524288s   driveoffset 524272s drive rootdev
volume swap
  plex org concat
#  b:      524007               281      swap
    sd len 524007s   driveoffset 265s drive rootdev
volume usr
  plex org concat
#  d:      4194304              1048576    4.2BSD     2048 16384 28512
    sd len 4194304s  driveoffset 1048560s drive rootdev
volume var
  plex org concat
#  e:      3142987              5242880    4.2BSD     2048 16384 28512
    sd len 3142987s  driveoffset 5242864s drive rootdev
```

The comments are the corresponding lines from the *bsdlabel* output. They show the corresponding values for size and offset. Run *vinum create* against this file, and confirm that you have the volumes */*, */usr* and */var*.

Next, ensure that you are set up to start Vinum with the new method. Ensure that you

have the following lines in */boot/loader.conf*, creating it if necessary:

```
vinum_load="YES"
vinum.autostart="YES"
```

Then reboot to single-user mode, start Vinum and run *fsck* against the volumes, using the `-n` option to tell *fsck* not to correct any errors it finds. You should see something like this:

```
# fsck -n -t ufs /dev/vinum/usr
** /dev/vinum/usr (NO WRITE)
** Last Mounted on /usr
** Phase 1 - Check Blocks and Sizes
** Phase 2 - Check Pathnames
** Phase 3 - Check Connectivity
** Phase 4 - Check Reference Counts
** Phase 5 - Check Cyl groups
35323 files, 314115 used, 718036 free (4132 frags, 89238 blocks, 0.4% fragmentation)
```

If there are any errors, they will probably be because you have miscalculated size or offset. You'll see something like this:

```
# fsck -n -t ufs /dev/vinum/usr
** /dev/vinum/usr (NO WRITE)
Cannot find file system superblock
/dev/vinum/usr: CANNOT FIGURE OUT FILE SYSTEM PARTITION
```

You need to do this in single-user mode because the volumes are shadowing file systems, and it's normal for open file systems to fail *fsck*, since some of the state is in buffer cache.

If all is well, remount the root file system read-write:

```
# mount -u /
```

Then edit */etc/fstab* to point to the new devices. For this example, */etc/fstab* might initially contain:

```
# Device                Mountpoint      FStype  Options         Dump    Pass#
/dev/da0s4a             /               ufs     rw              1       1
/dev/da0s4b             none            swap    sw              0       0
/dev/da0s4e             /usr            ufs     rw              1       1
/dev/da0s4f             /var            ufs     rw              1       1
```

Change it to reflect the Vinum volumes:

```
# Device                Mountpoint      FStype  Options         Dump    Pass#
/dev/vinum/swap         none            swap    sw              0       0
/dev/vinum/root         /               ufs     rw              1       1
/dev/vinum/usr          /usr            ufs     rw              1       1
/dev/vinum/var          /var            ufs     rw              1       1
```

Then reboot again to mount the root file system from */dev/vinum/root*. You can also optionally remove all the UFS partitions *except the root partition*. The loader doesn't know about Vinum, so it must boot from the UFS partition.

Once you have reached this stage, you can add additional plexes to the volumes, or you can extend the plexes (and thus the size of the file system) by adding subdisks to the plexes, as discussed on page 229.

Recovering from drive failures

One of the purposes of Vinum is to be able to recover from hardware problems. If you have chosen a redundant storage configuration, the failure of a single component will not stop the volume from working. In many cases, you can replace the components without down time.

If a drive fails, perform the following steps:

1. Replace the physical drive.
2. Partition the new drive. Some restrictions apply:
   - If you have hot-plugged the drive, it must have the same ID, the Vinum drive must be on the same partition, and it must have the same size.
   - If you have had to stop the system to replace the drive, the old drive will not be associated with a device name, and you can put it anywhere. Create a Vinum partition that is at least large enough to take all the subdisks *in their original positions on the drive*. Vinum currently does not compact free space when replacing a drive. An easy way to ensure this is to make the new drive at least as large as the old drive.

     If you want to have this freedom with a hot-pluggable drive, you must stop Vinum and restart it.
3. If you have restarted Vinum, create a new drive. For example, if the replacement drive *data3* is on the physical partition */dev/da3s1h*, create a configuration file, say *configfile*, with the single line

   ```
   drive data3 device /dev/da3s1h
   ```

   Then enter:

   ```
   # vinum create configfile
   ```

4. Start the plexes that were down. For example, *vinum list* might show:

   ```
   vinum -> l -r test
   V test                  State: up       Plexes:       2 Size:         30 MB
   P test.p0             C State: up       Subdisks:     1 Size:         30 MB
   P test.p1             C State: faulty   Subdisks:     1 Size:         30 MB
   S test.p0.s0            State: up       PO:        0  B Size:         30 MB
   S test.p1.s0            State: obsolete PO:        0  B Size:         30 MB
   vinum -> start test.p1.s0
   Reviving test.p1.s0 in the background
   vinum -> vinum[295]: reviving test.p1.s0        this message appears after the prompt
   vinum[295]: test.p1.s0 is up                    (some time later)
   ```

Backup software

FreeBSD does not require special ‘‘backup software.’’ The base operating system supplies all the programs you need. The tape driver is part of the kernel, and the system includes a number of backup programs. The most popular are:

- *tar*, the *tape archiver*, has been around longer than anybody can remember. It is particularly useful for data exchange, since everybody has it. There are even versions of *tar* for Microsoft platforms. It’s also an adequate backup program.
- *cpio* is an alternative backup program. About its only advantage over *tar* is that it can read *cpio* format archives.
- *pax* is another alternative backup program. It has the advantage that it can also read and write *tar* and *cpio* archives.
- *dump* is geared more towards backups than towards archiving. It can maintain multiple levels of backup, each of which backs up only those files that have changed since the last backup of the next higher (numerically lower) level. It is less suited towards data exchange because its formats are very specific to BSD. Even older releases of FreeBSD cannot read dumps created under FreeBSD Release 5.
- *amanda*, in the Ports Collection, is another popular backup program.

Backup strategies are frequently the subject of religious wars. I personally find that *tar* does everything I want, but you’ll find plenty of people who recommend *dump* or *amanda* instead. In the following section, we’ll look at the basics of using *tar*. See the man page *dump(8)* for more information on *dump*.

tar

tar, the *tape archiver*, performs the following functions:

- Creating an *archive*, which can be a serial device such as a tape, or a disk file, from the contents of a number of directories.
- Extracting files from an archive.
- Listing the contents of an archive.

tar does not compress the data. The resulting archive is slightly larger than the sum of the files that it contains, since it also contains a certain amount of header information. You can, however, use the *gzip* program to compress a *tar* archive, and *tar* invokes it for you automatically with the `-z` option. The size of the resultant archives depends strongly on the data you put in them. JPEG images, for example, hardly compress at all, while text compresses quite well and can be as much as 90% smaller than the original file.

Creating a tar archive

Create an archive with the `c` option. Unlike most UNIX programs, *tar* does not require a hyphen (`-`) in front of the options. For example, to save your complete kernel source tree, you could write:

```
# tar cvf source-archive.tar /usr/src/sys
tar: Removing leading / from absolute path names in the archive.
usr/src/sys/
usr/src/sys/CVS/
usr/src/sys/CVS/Root
usr/src/sys/CVS/Repository
usr/src/sys/CVS/Entries
usr/src/sys/compile/
usr/src/sys/compile/CVS/
(etc)
```

The parameters have the following meaning:

- `cvf` are the options. `c` stands for *create* an archive, `v` specifies *verbose* operation (in this case, this causes *tar* to produce the list of files being archived), and `f` specifies that the next parameter is the name of the archive file.
- `source-archive.tar` is the name of the archive. In this case, it's a disk file.
- `/usr/src/sys` is the name of the directory to archive. *tar* archives all files in the directory, including most devices. For historical reasons, *tar* can't back up devices with minor numbers greater than 65536, and changing the format would make it incompatible with other systems.

The message on the first line (`Removing leading / ...`) indicates that, although the directory name was specified as `/usr/src/sys`, *tar* treats it as *usr/src/sys*. This makes it possible to restore the files into another directory at a later time.

You can back up to tape in exactly the same way:

```
# tar cvf /dev/nsa0 /usr/src/sys
```

There is a simpler way, however: if you don't specify a file name, *tar* looks for the environment variable `TAPE`. If it finds it, it interprets it as the name of the tape drive. You can make things a lot easier by setting the following line in the configuration file for your shell (*.profile* for *sh*, *.bashrc* for *bash*, *.login* for *csh* and *tcsh*):

```
TAPE=/dev/nsa0 export TAPE                  for sh and bash
setenv TAPE /dev/nsa0                       for csh and tcsh
```

After this, the previous example simplifies to:

```
# tar cv /usr/src/sys
```

always necessary to have a file system on the diskette—in fact, as we'll see, it can be a disadvantage. In addition, FreeBSD offers different kinds of file system, so it performs the two functions with different programs. In this section, we'll look at *fdformat*, which performs the low-level format. We'll look at how to create a *UFS* or Microsoft file system in the next section.

To format a diskette in the first floppy drive, */dev/fd0*, you would enter:

```
$ fdformat /dev/fd0
Format 1440K floppy '/dev/fd0'? (y/n): y
Processing ----------------------------------------
```

Each hyphen character (`-`) represents two tracks. As the format proceeds, the hyphens change to an **F** (Format) and then to **V** (Verify) in turn, so at the end the line reads

```
Processing VVVVVVVVVVVVVVVVVVVVVVVVVVVVVV done.
```

File systems on floppy

It's possible to use floppies as file systems under FreeBSD. You can create a *UFS* file system on a floppy just like on a hard disk. This is not necessarily a good idea: the *UFS* file system is designed for performance, not maximum capacity. By default, it doesn't use the last 8% of disk space, and it includes a lot of structure information that further reduces the space available on the disk. Here's an example of creating a file system, mounting it on the directory */A*, and listing the remaining space available on an empty 3½" floppy. Since release 5, FreeBSD no longer requires a partition table on a floppy, so you don't need to run *bsdlabel* (the replacement for the older *disklabel* program).

```
# newfs -O1 /dev/fd0                                  create a new file system
/dev/fd0: 1.4MB (2880 sectors) block size 16384, fragment size 2048
        using 2 cylinder groups of 1.00MB, 64 blks, 128 inodes.
super-block backups (for fsck -b #) at:
 32, 2080
# mount /dev/fd0 /A                                   mount the floppy on /A
# df -k /A                                            display the space available
Filesystem  1024-blocks     Used    Avail Capacity  Mounted on
/dev/fd0          1326        2     1218     0%    /A
```

Let's look at this in a little more detail:

- *newfs* creates the *UFS* file system on the floppy. We use the `-O1` flag to force the older UFS1 format, which leaves more usable space than the default UFS2.
- We have already seen *mount* on page 192. In this case, we use it to mount the floppy on the file system */A*.
- The *df* program shows the maximum and available space on a file system. By default, *df* displays usage in blocks of 512 bytes, an inconvenient size. In this example we use the `-k` option to display it in kilobytes. You can set a default block size via the environment variable `BLOCKSIZE`. If it had been set to 1024, we would see the same output without the `-k` option. See page 128 for more details of environment variables.

The output of *df* looks terrible! Our floppy only has 1218 kB left for normal user data, even though there is nothing on it and even *df* claims that it can really store 1326 kB. This is because *UFS* keeps a default of 8% of the space free for performance reasons. You can change this, however, with *tunefs*, the file system tune program:[1]

```
# umount /A                                        first unmount the floppy
# tunefs -m 0 /dev/fd0                             and change the minimum free to 0
tunefs: minimum percentage of free space changes from 8% to 0%
tunefs: should optimize for space with minfree < 8%
# tunefs -o space /dev/fd0                         change the optimization
tunefs: optimization preference changes from time to space
# mount /dev/fd0 /A                                mount the file system again
# df /A                                            and take another look
Filesystem  1024-blocks     Used    Avail Capacity  Mounted on
/dev/fd0          1326        2     1324     0%    /A
```

Still, this is a far cry from the claimed data storage of a Microsoft disk. In fact, Microsoft disks can't store the full 1.4 MB either: they also need space for storing directories and allocation tables. The moral of the story: only use file systems on floppy if you don't have any alternative.

Microsoft file systems

To create a Microsoft FAT12, FAT16 or FAT32 file system, use the *newfs_msdos* command:

```
$ newfs_msdos -f 1440 /dev/fd0
```

The specification `-f 1440` tells *newfs_msdos* that this is a 1.4 MB floppy. Alternatively, you can use the *mformat* command:

```
$ mformat A:
```

You can specify the number of tracks with the `-t` option, and the number of sectors with the `-s` option. To explicitly specify a floppy with 80 tracks and 18 sectors (a standard 3½" 1.44 MB floppy), you could enter:

```
$ mformat -t 80 -s 18 A:
```

mformat is one of the *mtools* that we look at in the next section.

Other uses of floppies

Well, you could take the disks out of the cover and use them as a kind of frisbee. But there is one other useful thing you can do with floppies: as an archive medium, they don't need a file system on them. They just need to be low-level formatted. For example, to write the contents of the current directory onto a floppy, you could enter:

---

1. To quote the man page: *You can tune a file system, but you can't tune a fish.*

```
$ tar cvfM /dev/fd0 .
./
.xfmrc
.x6530modkey
.uwmrc
.twmrc
.rnsoft
.rnlast
...etc
Prepare volume #2 for /dev/fd0 and hit return:
```

Note also the solitary dot (`.`) at the end of the command line. That's the name of the current directory, and that's what you're backing up. Note also the option `M`, which is short for `--multi-volume`. There's a very good chance that you'll run out of space on a floppy, and this option says that you have a sufficient supply of floppies to perform the complete backup.

To extract the data again, use *tar* with the `x` option:

```
$ tar xvfM /dev/fd0
./
.xfmrc
.x6530modkey
.uwmrc
...etc
```

See the man page *tar(1)* for other things you can do with *tar*.

Accessing Microsoft floppies

Of course, most of the time you get data on a floppy, it's not in *tar* format: it has a Microsoft file system on it. We've already seen the Microsoft file system type on page 190, but that's a bit of overkill if you just want to copy files from floppy. In this case, use the *mtools* package from the Ports Collection. *mtools* is an implementation of the MS-DOS programs *ATTRIB*, *CD*, *COPY*, *DEL*, *DIR*, *FORMAT*, *LABEL*, *MD*, *RD*, *READ*, *REN*, and *TYPE* under UNIX. To avoid confusion with existing utilities, the UNIX versions of these commands start with the letter `m`. They are also written in lower case. For example, to list the contents of a floppy and copy one of the files to the current (FreeBSD) directory, you might enter:

```
$ mdir                                          list the current directory on A:
 Volume in drive A is MESSED OS
 Directory for A:/

IO       SYS       33430    4-09-91   5:00a
MSDOS    SYS       37394    4-09-91   5:00a
COMMAND  COM       47845   12-23-92   5:22p
NFS              <DIR>     12-24-92  11:03a
DOSEDIT  COM        1728   10-07-83   7:40a
CONFIG   SYS         792   10-07-94   7:31p
AUTOEXEC BAT         191   12-24-92  11:10a
MOUSE            <DIR>     12-24-92  11:09a
      12 File(s)      82944 bytes free
$ mcd nfs                                       change to directory A:\NFS
$ mdir                                          and list the directory
 Volume in drive A is MESSED OS
 Directory for A:/NFS
```

```
.               <DIR>     12-24-92  11:03a
..              <DIR>     12-24-92  11:03a
HOSTS              5985   10-07-94   7:34p
NETWORK  BAT        103   12-24-92  12:28p
DRIVES   BAT         98   11-07-94   5:24p
...and many more
      51 File(s)      82944 bytes free
$ mtype drives.bat                              type the contents of DRIVES.BAT
net use c: presto:/usr/dos
c:
cd \nfs
# net use f: porsche:/dos
# net use g: porsche:/usr
$ mcopy a:hosts .                               copy A:HOSTS to local UNIX directory
Copying HOSTS
$ ls -l hosts                                   and list it
-rw-rw-rw-   1 root     wheel        5985 Jan 28 18:04 hosts
```

You must specify the drive letter to *mcopy*, because it uses this indication to decide whether the file name is a UNIX or a Microsoft file name. You can copy files from FreeBSD to the floppy as well, of course.

A word of warning. UNIX uses a different text data format from Microsoft: in UNIX, lines end with a single character, called **Newline**, and represented by the characters `\n` in the C programming language. It corresponds to the ASCII character **Line Feed** (represented by `^J`). Microsoft uses two characters, a **Carriage Return** (`^M`) followed by a **Line Feed**. This unfortunate difference causes a number of unexpected compatibility problems, since both characters are usually invisible on the screen.

In FreeBSD, you won't normally have many problems. Occasionally a program complains about non-printable characters in an input line. Some, like *Emacs*, show them. For example, Emacs shows our last file, *drives.bat*, like this:

```
net use c: presto:/usr/dos^M
c:^M
cd \nfs^M
# net use f: porsche:/dos^M
# net use g: porsche:/usr^M
```

This may seem relatively harmless, but it confuses some programs, including the C compiler and pagers like *more*, which may react in confusing ways. You can remove them with the `-t` option of *mcopy*:

```
$ mcopy -t a:drives.bat .
```

Transferring files in the other direction is more likely to cause problems. For example, you might edit this file under FreeBSD and then copy it back to the diskette. The results depend on the editor, but assuming we changed all occurrences of the word `porsche` to `freedom`, and then copied the file back to the diskette, Microsoft might then find:

```
C:> type drives.bat
net use c: presto:/usr/dos
                        c:
                          cd \nfs
                                 # net use f: freedom:/dos
                                                          # net use g: freedom:/usr
```

This is a typical result of removing the **Carriage Return** characters. The `-t` option to *mcopy* can help here, too. If you use it when copying *to* a Microsoft file system, it reinserts the **Carriage Return** characters.

look at the differences we need for the ISP *example.net*. In the middle of the Internet, things are even more extreme. There may be dozens of interfaces, and the choice of a route for a particular address may be much more complicated. In such an environment, two problems occur:

- The concept of a default route no longer has much significance. If each interface carries roughly equal traffic, you really need to specify the interface for each network or group of networks. As a result, the routing tables can become enormous.
- There are probably multiple ways to route packets destined for a specific system. Obviously, you should choose the best route. But what happens if it fails or becomes congested? Then it's not the best route any more. This kind of change happens frequently enough that humans can't keep up with it—you need to run *routing software* to manage the routing table.

Adding routes automatically

FreeBSD comes with all the currently available routing software, primarily the daemon *routed*. The newer *gated* used to be included as well, but it is no longer available for free. It is available from *http://www.nexthop.com/products/howto_order.shtml*. An alternative in the Ports Collection is *zebra*.

All these daemons have one thing in common: you don't need them. At any rate, you don't need them until you have at least two different connections to the Internet, and even then it's not sure. As a result, we won't discuss them here. If you do need to run routing daemons, read all about them in *TCP/IP Network Administration*, by Craig Hunt.

From our point of view, however, the routing protocols have one particular significance: the system expects the routing table to be updated automatically. As a result, it is designed to use the information supplied by the routing protocols to perform the update. This information consists of two parts:

- The address and netmask of the network (in other words, the address range).
- The address of the *gateway* that forwards data for this address range. The gateway is a directly connected system, so it also figures in the routing table.

Adding routes manually

As we saw in the previous section, the routing software uses only addresses, and not the interface name. To add routes manually, we have to give the same information.

The program that adds routes manually is called *route*. We need it to add routes to systems other than those to which we are directly connected.

To set up the routing tables for the systems connected only to our reference network (*freebie*, *presto*, *bumble* and *wait*), we could write:

```
# route add default gw
```

During system startup, the script */etc/rc.network* performs this operation automatically if you set the following variable in */etc/rc.conf*:

```
defaultrouter="223.147.37.5"    # Set to default gateway (or NO).
```

Note that we enter the address of the default router as an IP address, not a name. This command is executed before the name server is running. We can't change the sequence in which we start the processes: depending on where our name server is, we may need to have the route in place to access the name server.

On system *gw*, the default route goes via the *tun0* interface:

```
# defaultrouter="139.130.136.129" # Set to default gateway (or NO).
gateway_enable="YES"            # Set to YES if this host will be a gateway.
```

This is a PPP interface, so you don't need a `defaultrouter` entry; if you did, it would look like the commented-out entry above. On page 347 we'll see how PPP sets the default route.

We need to enable gateway functionality on this system, since it receives data packets on behalf of other systems. We'll look at this issue in more depth on page 313.

ISP's route setup

At the ISP site, things are slightly more complicated than at *example.org*. Let's look at the gateway machine *free-gw.example.net*. It has three connections, to the global Internet, to *example.org* and to another network, *biguser.com* (the network serviced by interface *ppp0*). To add the routes requires something like the following commands:

```
# route add default 139.130.237.65                igw.example.net
# route add -net 223.147.37.0 139.130.136.133     gw.example.org
# route add -net 223.147.38.0 -iface ppp0         local ppp0 interface
```

The first line tells the system that the default route is via *gw.example.org*. The second shows that the network with the base IP address `223.147.37.0` (*example.org*) can be reached via the gateway address 139.130.136.133, which is the remote end of the PPP link connected via *ppp3*. In the case of *biguser.com*, we don't know the address of the remote end; possibly it changes every time it's connected. As a result, we specify the name of the interface instead: we know it's always connected via *ppp0*.

To solve this problem, execute the agent in your current environment with *eval*, then run *ssh-add*:

```
$ eval 'ssh-agent'
$ ssh-add
Enter passphrase for /home/grog/.ssh/id_rsa: (enter the passphrase)
Identity added: /home/grog/.ssh/id_rsa (/home/grog/.ssh/id_rsa)
Identity added: /home/grog/.ssh/id_dsa (/home/grog/.ssh/id_dsa)
Identity added: /home/grog/.ssh/identity (grog@zaphod.example.org)
```

You can use *ssh-add*'s `-l` flag to list which keys the authentication agent currently knows about:

```
$ ssh-add -l
1024 02:20:1d:50:78:c5:7c:56:7b:1d:e3:54:02:2c:99:76 grog@zaphod.example.org (RSA1)
1024 95:d5:01:ca:90:04:7d:84:f6:00:32:7a:ea:a6:57:2d /home/grog/.ssh/id_rsa (RSA)
1024 53:53:af:22:87:07:10:e4:5a:2c:21:31:ec:29:1c:5f /home/grog/.ssh/id_dsa (DSA)
```

If you're using a Bourne-style shell such as *bash*, you can automate a lot of this by putting the following commands in your *.bashrc* or *.profile* file:

```
if      tty > /dev/null; then
        ssh-add -l > /dev/null
        if [ $? -ne 0 ]; then
                eval 'ssh-agent'
        fi
fi
```

This first uses the *tty* command to check if this is an interactive shell, then checks if you already have an authentication agent. If it doesn't, it starts one. Don't start a new authentication agent if you already have one: you'd lose any keys that the agent already knows. This script doesn't add keys, because this requires your intervention and could be annoying if you had to do it every time you start a shell.

Setting up X to use ssh

If you work with X, you have the opportunity to start a large number of concurrent *ssh* sessions. It would be annoying to have to enter keys for each session, so there's an alternative method: start X with an *ssh-agent*, and it will pass the information on to any *xterm*s that it starts. Add the following commands to your *.xinitrc*:

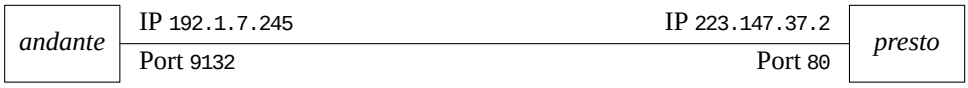
```
eval 'ssh-agent'
ssh-add < /dev/null
```

When you run *ssh-add* in this manner, without an input file, it runs a program to prompt for the passphrase. By default it's */usr/X11R6/bin/ssh-askpass*, but you can change it by setting the SSH_ASKPASS environment variable. */usr/X11R6/bin/ssh-askpass* opens a window and prompts for a passphrase. From then on, anything started under the X session will automatically inherit the keys.

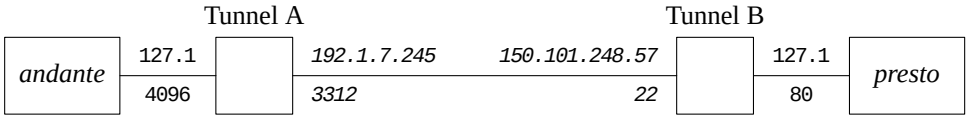
ssh tunnels

Tunneling is a technique for encapsulating an IP connection inside another IP connection. Why would you want to do that? One reason is to add encryption to an otherwise unencrypted connection, such as *telnet* or *POP*. Another is to get access to a service on a system that does not generally supply this service to the Internet.

Let's consider using *http* first. Assume you are travelling, and you want to access your private web server back home. Normally a connection to the `http` port of *presto.example.com* might have the following parameters:



But what if the server is firewalled from the global Internet, so you can't access it directly? That's when you need the *ssh* tunnel. The *ssh* tunnel creates a local connection at each end and a separate secure connection across the Internet:



The *ssh* connection is shown in *`fixed italic`* font. It looks just like any other *ssh* connection. The differences are the local connections at each end: instead of talking to presto port 80 (`http`), you talk to port 4096 on your local machine. Why 4096? It's your choice; you can use any port above 1024. If you're on *andante*, you can set up this tunnel with the command:

```
$ ssh -L 4096:presto.example.org:80 presto.example.org
```

To do the same thing from the *presto* end, you'd set up a *reverse tunnel* with the `-R` option:

```
$ ssh -R 4096:presto.example.org:80 andante.example.org
```

These commands both set up a tunnel from port 4096 on *andante* to port 80 on the host *presto.example.org*. You still need to supply the name of the system to connect to; it doesn't have to be the same. For example, you might not be able to log in to the web server, but you could access your machine back home, and it has access to the web server. In this case, you could connect to your machine at home:

```
$ ssh -L 4096:presto.example.org:80 freebie.example.org
```

In addition to setting up the tunnel, *ssh* can create a normal interactive session. If you don't want this, use the `-f` option to tell *ssh* to go into the background after authentication. You can also specify a command to execute, but this is no longer

necessary for protocol version 2. If you don't want to execute a command, use the `-N` option:

```
$ ssh -L 4096:presto.example.org:80 presto.example.org -f -N
```

If you're running protocol version 1, you can use *sleep* with an appropriately long timeout, in this example 1 hour:

```
$ ssh -L 4096:presto.example.org:80 presto.example.org -f sleep 3600
```

Tunneling X

Running X clients on the remote machine is special enough that *ssh* provides a special form of tunneling to deal with it. To use it, you must tell *ssh* the location of an *.Xauthority* file. Do this by adding the following line to the file *~/.ssh/environment*:

```
XAUTHORITY=/home/yourname/.Xauthority
```

The name must be in fully qualified form: *ssh* does not understand the shortcut *~/* to represent your home directory. You don't need to create *~/.Xauthority*, though: *ssh* can do that for you.

Once you have this in place, you can set up X tunneling in two different ways. To start it from the command line, enter something like:

```
$ ssh -X -f website xterm
```

As before, the `-f` option tells *ssh* to go into the background. The `-X` option specifies X tunneling, and *ssh* runs an *xterm* on the local machine. The `DISPLAY` environment variable points to the (remote) local host:

```
$ echo $DISPLAY
localhost:13.1
```

Other uses of tunnels

Tunneling has many other uses. Another interesting one is bridging networks. For example, *http://unix.za.net/gateway/documentation/networking/vpn/fbsd.html* describes how to set up a VPN (Virtual Private Network) using User PPP and an *ssh* tunnel.

Configuring ssh

It can be a bit of a nuisance to have to supply all these parameters to *ssh*, but you don't have to: you can supply information for frequently accessed hosts in a configuration file. On startup, *ssh* checks for configuration information in a number of places. It checks for them first in the command-line options, then in you configuration file *~/.ssh/config*, and finally in the system-wide configuration file */etc/ssh/ssh_config*. The way it treats

duplicate information is pretty much the opposite of what you'd expect: unlike most other programs, options found in a configuration file read in later do *not* replace the options found in an earlier file. Options on the command line replace those given in configuration files.

In practice, such conflicts happen less often than you might expect. The file */etc/ssh/ssh_config*, the main configuration file for the system, normally contains only comments, and by default you don't even get a local *~/.ssh/config*.

ssh_config can contain a large number of options. They're all described in the man page *ssh_config(8)*, but it's worth looking at some of the more common ones. In this section we'll look at some of the more common configuration options.

- The entry `Host` is special: the options that follow, up to the end of the file or the next following `Host` argument, relate only to hosts that match the arguments on the `Host` line.
- Optionally, *ssh* can compress the data streams. This can save a lot of traffic, but it can also increase CPU usage, so by default it is disabled. You can do this by passing the `-C` flag to *ssh*, but you can also do so by setting `Compression yes` in the configuration file.
- You can *escape* out of an *ssh* session to issue commands to *ssh* with the `EscapeChar`. By default it's the *tilde* character, `~`. Other programs, notably *rlogin*, use this character as well, so you may want to change it. You can set this value from the *ssh* command line with the `-e` option.
- To forward an X11 connection, as shown above, you can also set the `ForwardX11` variable to `yes`. This may be useful if you frequently access a remote machine and require X forwarding. This also sets the `DISPLAY` environment variable correctly to go over the secure channel.
- By default, *ssh* sends regular messages to the remote *sshd* server to check if the remote system has gone down. This can cause connections to be dropped on a flaky connection. Set the `KeepAlive` option to `no` to disable this behaviour.
- Use the `LocalForward` parameter to set up a tunnel. The syntax is similar to that of the `-L` option above: on *andante*, instead of the command line:

  ```
  $ ssh -L 4096:presto.example.org:80 presto.example.org
  ```

  you would put the following in your *~/.ssh/config*:

  ```
  host presto.example.org
  LocalForward 4096 presto.example.org:80
  ```

  Note that the first port is separated from the other two parameters by a space, not a colon.

- Similarly, you can set up a reverse tunnel with the `RemoteForward` parameter. On *presto*, instead of the command line:

  ```
  $ ssh -R 4096:presto.example.org:80 andante.example.org
  ```

  you would put the following in your *~/.ssh/config*:

  ```
  host andante.example.org
  RemoteForward 4096 presto.example.org:80
  ```

- By default, *ssh* uses password authentication if it can't negotiate a key pair. Set `PasswordAuthentication` to `no` if you don't want this.

- Normally *ssh* connects to the server on port 22 (`ssh`). If the remote server uses a different port, specify it with the `Port` keyword. You can also use the `-p` option on the *ssh* command line.

- By default, *ssh* attempts to connect using protocol 2, and if that doesn't work, it tries to connect using protocol 1. You can override this default with the `Protocol` keyword. For example, to reverse the default and try first protocol 1, then protocol 2, you would write:

  ```
  Protocol        1,2
  ```

- By default, *ssh* refuses to connect to a known host if its key fingerprint changes. Instead, you must manually remove the entry for the system from the *~/.ssh/known_hosts* or *~/.ssh/known_hosts2* file. This can indicate that somebody is faking the remote machine, but more often it's because the remote machine has really changed its host key, which it might do at every reboot. If this gets on your nerves, you can add this line to your configuration file:

  ```
  StrictHostKeyChecking  no
  ```

  This doesn't stop the warnings, but *ssh* continues:

  ```
  @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
  @    WARNING: REMOTE HOST IDENTIFICATION HAS CHANGED!     @
  @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
  IT IS POSSIBLE THAT SOMEONE IS DOING SOMETHING NASTY!
  Someone could be eavesdropping on you right now (man-in-the-middle attack)!
  It is also possible that the DSA host key has just been changed.
  The fingerprint for the DSA key sent by the remote host is
  95:80:4c:fb:cc:96:1b:36:c5:c9:2b:cb:d1:d4:16:68.
  Please contact your system administrator.
  Add correct host key in /home/grog/.ssh/known_hosts2 to get rid of this message.
  Offending key in /home/grog/.ssh/known_hosts2:39
  ```

- *ssh* assumes that your user name on the remote system is the same as the name on the local system. If that's not the case, you can use the `User` keyword to specify the remote user name. Alternatively, you can use the format:

  ```
  $ ssh newuser@remotehost.org
  ```

Summary of files in ~/.ssh

In addition to the files we have discussed, you will find two other files in the *~/.ssh* directory:

- *known_hosts* contains the key fingerprints of all hosts to which you have connected. The example on page 419 shows how *ssh* adds a key.
- *random_seed* is a seed used to generate the keys.

In summary, then, you can expect the following files in your *~/.ssh*:

```
drwx------  2 grog  grog   512 Jan 18 21:04 .                  directory
-rw-r--r--  1 grog  grog  1705 Oct 26  1999 authorized_keys     keys
-rw-r--r--  1 grog  grog   844 Jan 27 22:18 authorized_keys2    keys, Version 2 only
-rw-r--r--  1 grog  grog    25 Oct 20 01:35 environment         environment for sshd
-rw-------  1 grog  grog   736 Jul 19 15:40 id_dsa              DSA private key
-rw-r--r--  1 grog  grog   611 Jul 19 15:40 id_dsa.pub          DSA public key
-rw-------  1 grog  grog   951 Jul 19 15:40 id_rsa              RSA private key
-rw-r--r--  1 grog  grog   231 Jul 19 15:40 id_rsa.pub          RSA public key
-rw-------  1 grog  grog   536 Jul 19 15:39 identity           RSA1 private key
-rw-r--r--  1 grog  grog   340 Jul 19 15:39 identity.pub       RSA1 public key
-rw-------  1 grog  grog  1000 Jul 25  1999 known_hosts        list of known hosts
-rw-------  1 grog  grog   512 Jul 25  1999 random_seed        for key generation
```

Note particularly the permissions and the ownership of the files and the directory itself. If they are wrong, *ssh* won't work, and it won't tell you why not. In particular, the directory must not be group writeable.

Troubleshooting ssh connections

A surprising number of things can go wrong with setting up *ssh* connections. Here are some of the more common ones:

- After some delay, you get the message:

  ```
  ssh: connect to address 223.147.37.76 port 22: Operation timed out
  ```

  This probably means that the remote host is down, or that you can't reach it due to network problems.

- You get the message:

  ```
  ssh: connect to address 223.147.37.65 port 22: Connection refused
  ```

  This means that the remote host is up, but no *sshd* is running.

- You have set up keys, but you still get a message asking for a password.

  This can mean a number of things: your *ssh-agent* isn't running, you haven't added the keys, the other end can't find them, or the security on the keys at the other end is incorrect. You can check the first two like this:

```
$ ssh-add -l
Could not open a connection to your authentication agent.
```

This message means that you haven't run *ssh-agent*. Do it like this:

```
$ eval 'ssh-agent'
Agent pid 95180
$ ssh-add -l
The agent has no identities.
$ ssh-add
Enter passphrase for /home/grog/.ssh/id_rsa: no echo
Identity added: /home/grog/.ssh/id_rsa (/home/grog/.ssh/id_rsa)
Identity added: /home/grog/.ssh/id_dsa (/home/grog/.ssh/id_dsa)
Identity added: /home/grog/.ssh/identity (grog@freebie.lemis.com)
$ ssh-add -l
1024 02:20:1d:50:78:c5:7c:56:7b:1d:e3:54:02:2c:99:76 grog@zaphod.example.org (RSA1)
1024 95:d5:01:ca:90:04:7d:84:f6:00:32:7a:ea:a6:57:2d /home/grog/.ssh/id_rsa (RSA)
1024 53:53:af:22:87:07:10:e4:5a:2c:21:31:ec:29:1c:5f /home/grog/.ssh/id_dsa (DSA)
```

In this case, all three keys are set correctly. If you have, say, only an RSA1 (protocol Version 1) key, and the other end doesn't support protocol Version 1, *ssh* will ask for a password.

- You get a message like this:

```
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@    WARNING: REMOTE HOST IDENTIFICATION HAS CHANGED!     @
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
IT IS POSSIBLE THAT SOMEONE IS DOING SOMETHING NASTY!
Someone could be eavesdropping on you right now (man-in-the-middle attack)!
It is also possible that the DSA host key has just been changed.
The fingerprint for the DSA key sent by the remote host is
95:80:4c:fb:cc:96:1b:36:c5:c9:2b:cb:d1:d4:16:68.
Please contact your system administrator.
Add correct host key in /home/grog/.ssh/known_hosts2 to get rid of this message.
Offending key in /home/grog/.ssh/known_hosts2:39
```

There are two possible reasons for this message. As the message states, one is that somebody is trying to intercept the connection, and the other one is that the remote system has changed its host key. The latter is by far the more common. To fix this problem, you have two choices:

1. Edit your *~/.ssh/known_hosts2* file and remove references to the remote system. The message suggests changing line 39, but you might have more than one key for this system in this file. If one is wrong, there's a good chance that any others will be too, so you should remove all references.
2. Add the following line to your *~/.ssh/config* file:

   ```
   StrictHostKeyChecking  no
   ```

   It doesn't remove the warning, but it allows you to connect anyway.

ssh includes debugging options that may help debug problems setting up connections. Use the `-v` option, up to three times, to get *ssh* to display largely undocumented information about what is going on. The output is pretty verbose; with three `-v` options you get nearly 200 lines of information.

telnet

As mentioned above, *telnet* is an older, unencrypted program that connects to a shell on a remote system. You might find it of use when connecting to a system that doesn't have *ssh*. Be very careful not to use valuable passwords, since they are transmitted in the clear. Apart from that, you use it pretty much in the same way as *ssh*:

```
$ telnet freebie
Trying 223.147.37.1...
Connected to freebie.example.org.
Escape character is '^]'.
login: grog
Password: (no echo)

FreeBSD/i386 (wantadilla.example.org) (ttypj)

Last login: Mon Oct 14 17:51:57 from sydney.example.org
Copyright (c) 1980, 1983, 1986, 1988, 1990, 1991, 1993, 1994
        The Regents of the University of California.  All rights reserved.

FreeBSD 5.0-RELEASE (FREEBIE) #0: Tue Dec 31 19:08:24 CST 2002

You have new mail.
If I have seen farther than others, it is because I was standing on the
shoulders of giants.
                -- Isaac Newton

In the sciences, we are now uniquely privileged to sit side by side
with the giants on whose shoulders we stand.
                -- Gerald Holton

If I have not seen as far as others, it is because giants were standing
on my shoulders.
                -- Hal Abelson

In computer science, we stand on each other's feet.
                -- Brian K. Reid
$ tty
/dev/ttyp9
$
```

Once you get this far, you are connected to the machine in an almost identical manner as if you were directly connected. This is particularly true if you are running X. As the output of the *tty* command shows, your ''terminal'' is a *pseudo-tty* or *pty* (pronounced ''pity''). This is the same interface that you will have with an *xterm*.

It's worth looking in more detail at how the connection is established:

- The first line (*Trying...*) appears as soon as *telnet* has resolved the IP address.
- The next three lines appear as soon as it has a reply from the other end. At this point, there can be a marked delay before *telnet* continues. *telnet* performs a reverse DNS lookup to find the name of your system. If you get a delay here, it could be an indication that your reverse lookup is not working correctly. After DNS times out, it will continue normally, but the delay is a nuisance.

```
$ ftp ftp.tu-darmstadt.de
Connected to ftp.tu-darmstadt.de.
220 rs3.hrz.th-darmstadt.de FTP server (Version 4.1) ready.
331 Password required for grog.
530 Login incorrect.
Login failed.
Remote system type is UNIX.
Using binary mode to transfer files.
ftp>
```

This error message is not very obvious: although you're not logged in, you still get the same prompt, and *ftp* produces enough verbiage that it's easy to oversee that the login attempt failed. To complete the login, use the *user* command:

```
ftp> user ftp
331 Guest login ok, send ident as password.
Password:                          username does not echo
230 Guest login ok, access restrictions apply.
```

sftp

sftp is yet another *ssh*-based program. It's designed to be as compatible as possible with *ftp*, so you use it in exactly the same manner. As with other *ssh*-related commands, you need to authenticate in an *ssh*-specific manner. In addition, it has an *exec* command, which allows you to run programs on the remote machine.

To use *sftp*, the remote machine must be able to run the *sftp-server* server. It is normally started from *sshd*. See page 454 for more details.

rsync

Frequently you want to keep identical copies of files on different machines. You can copy them, of course, but if there are only small changes in large files, this can be relatively inefficient. You can perform this task more efficiently with *rsync*, which is designed to keep identical copies of files on two different systems and to optimize network bandwidth while doing so. It's in the Ports Collection. Install in the normal manner:

```
# cd /usr/ports/net/rsync
# make install
```

By default, *rsync* uses *ssh* to perform the transfer, so you need to have *ssh* configured correctly. In particular, you should be using *ssh-agent* authentication.

You can use *rsync* like *scp*: the syntax is compatible up to a point. For example, you could copy a file from a remote system with:

```
$ rsync presto:/var/log/messages prestomessages
```

You don't need to install *rsync* just for that, of course: you can do exactly the same thing with *scp*. *rsync* has one advantage over *scp*, however, even in this case. The first time you copy the file, there's no difference. But files like */var/log/messages* grow at the end, and the rest doesn't change. That's an ideal situation for *rsync*: it uses an algorithm that recognizes common parts of files (not necessarily at the beginning) and optimizes the transfer accordingly. The first time you run the program, you might see:

```
$ rsync -v /var/log/messages freebie:/var/tmp
messages
wrote 80342 bytes  read 36 bytes  53585.33 bytes/sec
total size is 80255  speedup is 1.00
$ rsync -v /var/log/messages freebie:/var/tmp
messages
wrote 535 bytes  read 726 bytes  840.67 bytes/sec
total size is 80255  speedup is 63.64
```

This example used the option `-v` to show details of what was transferred; otherwise you wouldn't see any output at all. The first time round, the entire file was copied, so there was no speedup. The second time, though, almost nothing needed to be copied, so the transfer was over 60 times as fast.

Copying directory hierarchies

rsync has a bewildering number of options for synchronizing directories. Consider the case where you maintain web pages locally, but your main web server is co-located somewhere else. After updating the local web pages, you can run a script to update the remote pages with commands like:

```
rsync -LHzav --exclude=RCS --exclude="*~" ~grog/public_html/* website:htdocs/grog
rsync -LHztpgov --exclude="*~" website:htdocs
```

The first *rsync* command synchronizes the local directory *~grog/public_html* to the remote directory *htdocs/grog* on the system *website*. It includes all subdirectories with the exception of the *RCS* directories. The second command synchronizes the top level web directory only, and not the subdirectories, many of which shouldn't be maintained on the remote site. In each case, files ending in ~ are excluded (these are normally *Emacs* backup files), and in the second case the *RCS* subdirectories are also excluded. Let's look more carefully at all those options:

- `-L` copies symbolic links (which the documentation refers to as ''soft links'') as separate files. If you don't include this option, symbolic links to files within the directory hierarchy will work, but links outside the hierarchy may be broken (depending on whether a file of that name exists on the destination system or not). In this example, a number of files are really located elsewhere, so it makes sense to copy them as files.

Mounting NFS file systems automatically

If you want to mount NFS files automatically at boot time, make an entry for them in the file */etc/fstab*. You can even do this if you don't necessarily want to mount them: just add the keyword `noauto`, and *mountall* will ignore them at boot time. The advantage is that you then just need to specify, say,

```
# mount /src
```

instead of:

```
# mount -s freebie:/src /src
```

See the description of */etc/fstab* on page 566 for more information.

NFS strangenesses

NFS mimics a local file system across the network. It does a pretty good job, but it's not perfect. Here are some things that you should consider.

No devices

NFS handles disk files and directories, but not devices. Actually, it handles devices too, but not the way you would expect.

In a UNIX file system, a device is more correctly known as a *device node*: it's an inode that *describes* a device in terms of its major and minor numbers (see page 195). The device itself is implemented by the device driver. NFS exports device nodes in UFS file systems, but it doesn't interpret the fact that these devices are on another system. If you refer to the devices, one of three things will happen:

- If a driver for the specified major number exists on your local system, and the devices are the same on both systems, you will access the local device. Depending on which device it is, this could create some subtle problems that could go undetected for quite a while.
- If a driver for the specified major number exists on your local system, and the devices are different on the two systems, you will still access the local device with the same major and minor numbers, if such a device exists. The results could be very confusing.
- If no driver for the specified major number exists on your local system, the request will fail. This can still cause considerable confusion.

If the NFS server system runs *devfs*, the device nodes are not exported. You won't see anything unless there are leftover device nodes from before the time of migration to *devfs*.

Just one file system

NFS exports file systems, not directory hierarchies. Consider the example on page 444. *presto* has mounted both *freebie:/* and *freebie:/usr*. If it were just to mount *freebie:/*, we would see the directory */freebie/usr*, but it would be empty.

Things can get even stranger: you can mount a remote file system on a directory that is not empty. Consider the following scenario:

- You install FreeBSD on system *freebie*. In single-user mode, before mounting the other file systems, you create a directory */usr/bin* and a file */usr/bin/vi*. Since the */usr* file system isn't mounted, this file goes onto the root file system.
- You go to multi-user mode and mount the other file systems, including the file system for */usr*. You can no longer see the */usr/bin/vi* you put there in single-user mode. It hasn't gone away, it's just masked.
- On *presto*, you mount the file system *freebie:/* on */freebie*. If you list the contents of the directory */freebie/usr*, you will see the original file *vi*, and not the contents that the users on *freebie* will see.

Multiple monitors across multiple servers

We saw above that a server can handle multiple monitors, and a system can handle multiple servers. One problem with multiple monitors is that most computers can only handle a small number of display boards: a single AGP board and possibly a number of PCI boards. But PCI boards are difficult to find nowadays, and they're slower and have less memory.

If you have a number of machines located physically next to each other, you have the alternative of running X on each of them and controlling everything from one keyboard and mouse. You do this with the *x11/x2x* port. For example: *freebie*, *presto* and *bumble* have monitors next to each other, and *presto* has two monitors. From left to right they are *freebie:0.0*, *presto:0.0*, *presto:0.1* and *bumble:0.0*. The keyboard and mouse are connected to *presto*. To incorporate *freebie:0.0* and *bumble:0.0* in the group, enter these commands on *presto*:

```
$ DISPLAY=:0.0 x2x -west -to freebie:0 &
$ DISPLAY=:0.1 x2x -east -to bumble:0 &
```

After this, you can move to the other machines by moving the mouse in the corresponding direction. It's not possible to continue to a further machine, but it is possible to connect in other directions (*north* and *south*) from each monitor on *presto*, which in this case would allow connections to at least six other machines. Before that limitation becomes a problem, you need to find space for all the monitors.

Stopping X

To stop X, press the key combination **Ctrl**-**Alt**-**Backspace**, which is deliberately chosen to resemble the key combination **Ctrl**-**Alt**-**Delete** used to reboot the machine. **Ctrl**-**Alt**-**Backspace** stops X and returns you to the virtual terminal in which you started it. If you run from *xdm*, it redisplays a login screen.

```
.............................done
.
```

After that, nothing appears on the screen for quite some time. In fact, the boot is proceeding normally, and the next thing you see is a login prompt.

Configuring the machine

Setting up a diskless machine is not too difficult, but there are some gotchas:

- Currently, locking across NFS does not work properly. As a result, you may see messages like this:

  ```
  Dec 11 14:18:50 bumble sm-mta[141]: NOQUEUE: SYSERR(root): cannot flock(/var/run/
  sendmail.pid, fd=6, type=2, omode=40001, euid=0): Operation not supported
  ```

  One solution to this problem is to mount */var* as an MD (memory) file system. This is what currently happens by default, though it's subject to change: at startup, when the system detects that it is running diskless (via the sysctl `vfs.nfs.diskless_valid`), it invokes the configuration file */etc/rc.diskless1*. This file in turn causes the file */etc/rc.diskless2* to be invoked later in the startup procedure. Each of these files adds an MD file system. In the course of time, this will be phased out and replaced by the traditional configuration via */etc/fstab*, but at the moment this file has no provision for creating MD file systems.

  You should probably look at these files carefully: they may need some tailoring to your requirements.

- It is currently not possible to add swap on an NFS file system. *swapon* (usually invoked from the startup scripts) reports, incorrectly:

  ```
  Dec 11 14:18:46 bumble savecore: 192.109.197.82:/src/nodisk/swap/bumble: No such
   file or directory
  ```

  This, too, will change; in the meantime, it *is* possible to mount swap on files, even if they are NFS mounted, but not on the NFS file system itself. This means that the first of the following entries in */etc/fstab* will not work, but the second will:

  ```
  192.109.197.82:/src/nodisk/swap/bumble  none swap  sw          0       0
  /src/nodisk/swap/bumble                 none swap  sw          0       0
  echunga:/src            /src            nfs     rw          0       0
  ```

  The reason here is the third line: */src/nodisk/swap/bumble* is NFS mounted, so this is a swap-to-file situation. For this to work, you may have to add the following line at the end of your */etc/rc.diskless2*:

  ```
  swapon -a
  ```

This is because the standard system startup mounts swap before mounting additional NFS file systems. If you place the swap file on the root file system, it will still work, but frequently you will want the root file system to be read-only to be able to share it between several machines.

- If the machine panics, it's not possible to take a dump, because you have no disk. The only alternative would be a kernel debugger.

Sharing system files between multiple machines

In many cases, you may have a number of machines that you want to run diskless. If you have enough disk (one image for each machine), you don't have anything to worry about, but often it may be attractive to share the system files between them. There are a lot of things to consider here:

- Obviously, any changeable data specific to a system can't be shared.
- To ensure that things don't change, you should mount shared resources read-only.
- Refer to Table 32-1 on page 594 for an overview of FreeBSD installed directories. Of these directories, only */etc* and */usr/local/etc* must be specific for a particular system, though there are some other issues:
  - Installing ports, for example, will install ports for all systems. That's not necessarily a bad thing, but if you have two systems both installing software in the same directory, you can expect conflicts. It's better to designate one system, possibly the host with the disk, to perform these functions.
  - If you share */boot* and make some configuration changes, the options will apply to all systems.
  - When building system software, you can use the same */usr/src* and */usr/obj* directories as long as all systems maintain the same release of FreeBSD. You can even have different kernels: each kernel build directory carries the name of the configuration file, which by convention matches the name of the system.

The big problem is */etc*. In particular, */etc/rc.conf* contains information like the system name. One way to handle this is to have a separate */etc* directory for each system. This may seem reasonable, because */etc* is only about 1.5 MB in size. In fact, this implies mounting the entire root file system with the other top-level directories, and that means more like 60 MB.

/etc/crontab

/etc/crontab describes the jobs to be performed by *cron* on behalf of the system. You don't have to use this file at all; you can use each user's *crontab* files instead. Note that this file has a slightly different format from the other *crontab* files. A user's *crontab* contains entries like this:

```
0       0       *       *       *       /home/grog/Scripts/rotate-log
```

This line runs the script */home/grog/Scripts/rotate-log* at midnight every day. If you put this entry into */etc/crontab*, you need to tell *cron* which user to run it as. Do this by putting the name of the user before the command:

```
0       0       *       *       *       grog    /home/grog/Scripts/rotate-log
```

See page 151 for more details about *cron*.

/etc/csh.cshrc, /etc/csh.login, /etc/csh.logout

These are default initialization files for *csh*. See the man page *csh(1)* for more details.

/etc/dhclient.conf

/etc/dhclient.conf describes the client side of DHCP services. Normally it's empty. We discussed *dhcp* on 302.

/etc/disktab

/etc/disktab contains descriptions of disk geometries for *disklabel*. This is almost obsolete.

/etc/ftpusers

/etc/ftpusers is a list of users who are *not* allowed to connect to this system using *ftp*. It's a strong contender for the prize for the worst-named file in the system.

/etc/hosts

For a small network, especially if you're not permanently connected to the Internet, you have the option of placing the addresses of the systems you want to talk to in a file called */etc/hosts*. This file is simply a list of IP addresses and host names, for example:

```
# Local network host addresses
#
# loopback address for all systems
127.1 loopback local localhost
###### domain example.com.
#
223.147.37.1   freebie freebie.example.org        # FreeBSD 3.0
223.147.37.2   presto.example.org presto          # 66 MHz 486 (BSD UNIX)
```

Before the days of DNS, this was the way to resolve IP addresses. It only works locally, and even there it's a pain to maintain: you need to propagate every update to every machine on the network. As we saw in Chapter 21, it's far preferable to run *named*, even if you're not connected to the Internet.

/etc/hosts.equiv

/etc/hosts.equiv is a list of hosts whose users may use *rsh* to access this system without supplying a password. *rsh* is now obsolete, so it's unlikely you'll need to change this file. See the description of *ssh* on page 419 for a replacement.

/etc/hosts.lpd

/etc/hosts.lpd is a list of hosts that can use the *lpd* spooler on this system.

/etc/inetd.conf

/etc/inetd.conf is the configuration file for *inetd*, the Internet daemon. It dates back to the original implementation of TCP/IP in 4.2BSD, and the format is the same for all versions of UNIX. We have looked at various modifications to this file throughout the network part of the book. See the index (*inetd.conf*) and the man page *inetd.conf(5)* for further details. FreeBSD now disables all services by default to limit security exposures, so there's a good chance you'll have to edit this file.

/etc/login.access

/etc/login.access is a file that limits remote access by individual users. We don't look at it in more detail here.

/etc/login.conf

/etc/login.conf describes user parameters set at login time.

In UNIX tradition, `root` has been the owner of the universe. This is rather primitive, and the 4.3BSD Net/2 relase introduced *login classes*, which determine session accounting, resource limits and user environment settings. Many programs use the database described in */etc/login.conf* to set up a user's login environment and to enforce policy, accounting and administrative restrictions. The login class database also provides the means to authenticate users to the system and to choose the type of authentication.

When creating a user, you may optionally enter a class name, which should match an entry in */etc/login.conf*—see page 146 for more details. If you don't, the system uses the entry `default` for a non-root user. For the root user, the system uses the entry `root` if it is present, and `default` otherwise.

The structure of the login configuration database is relatively extensive. It describes a number of parameters, many of which can have two values: a *current* value and a *maximum* value. On login, the system sets the values to the `-cur` (current) value, but the user may, at his option, increase the value to the `-max` (maximum) value. We'll look at the `default` entry for an example.

The solution to this issue is called *mergemaster*, a script that helps you to upgrade the configuration files. We'll look at it in more detail below, but at this point you should know that you need to run it with the `-p` (*pre-build*) option:

```
# mergemaster -p
```

As we've seen in table 32-1, the `installworld` target changes a number of directories. Sometimes, though, it leaves old binaries behind: it doesn't remove anything that it doesn't replace. The result can be that you end up using old programs that have long passed their use-by date. One solution to this problem is to look at the last modification date of each program in the directories. For example, if you see:

```
$ ls -lrt /usr/sbin
-r-xr-xr-x  1 root  wheel       397 Jul 14 11:36 svr4
-r-xr-xr-x  1 root  wheel       422 Jul 14 11:29 linux
-r-xr-xr-x  1 root  wheel    142080 Jul 13 17:20 sshd
...
-r-xr-xr-x  1 root  wheel     68148 Jul 13 17:16 uuchk
-r-xr-xr-x  1 root  wheel      6840 Jan  5  2002 ispppcontrol
-r-xr-xr-x  1 root  wheel     27996 Apr 21  2001 k5stash
-r-xr-xr-x  1 root  wheel     45356 Apr 21  2001 ktutil
-r-xr-xr-x  1 root  wheel     11124 Apr 21  2001 kdb_util
-r-xr-xr-x  1 root  wheel      6768 Apr 21  2001 kdb_init
```

It's fairly clear that the files dated April 2001 have not just been installed, so they must be out of date. You can use a number of techniques to delete them; one might be:

```
# find . -mtime +10 | xargs rm
```

This command removes all files in the current directory (.) that are older than 10 days (`+10`). Of course, this method will only work if you haven't installed anything in these directories yourself. You shouldn't have done so; that's the purpose of the directory hierarchy */usr/local*, to ensure that you keep system files apart from ports and private files.

Be careful with */usr/lib*: a number of ports refer to libraries in this directory hierarchy, and if you delete them, the ports will no longer work. In general there's no problem with old libraries in */usr/lib*, unless they take up too much space, so you're safer if you don't clean out this directory hierarchy.

Note that you need to specify the `KERNCONF` parameter to all the targets relating to kernel builds.

Upgrading the kernel

There are two reasons for building a new kernel: it might be part of the upgrade process, which is what we'll look at here, or you may build a kernel from your current sources to add functionality to the system. We'll look at this aspect in Chapter 33.

One point to notice is that if you're upgrading from an older custom configuration file, you could have a lot of trouble. We'll see a strategy for minimizing the pain on page 616. In addition, when upgrading to FreeBSD Release 5 from an older release of FreeBSD, you need to install a file */boot/device.hints*, which you can typically copy from */usr/src/sys/i386/conf/GENERIC.hints*:

```
# cp /usr/src/sys/i386/conf/GENERIC.hints /boot/device.hints
```

See page 608 for more details.

When upgrading the kernel, you might get error messages like this one:

```
# config GENERIC
config: GENERIC:71: devices with zero units are not likely to be correct
```

Alternatively, you might get a clearer message:

```
# config GENERIC
../../conf/files: coda/coda_fbsd.c must be optional, mandatory or standard
Your version of config(8) is out of sync with your kernel source.
```

Apart from that, you might find that the kernel fails to link with lots of undefined references. This, too, could mean that the *config* program is out of synchronization with the kernel modules. In each case, build and install the new version of *config*:

```
# cd /usr/src/usr.sbin/config
# make depend all install clean
```

You need to *make clean* at the end since this method will store the object files in non-standard locations.

Upgrading the boot files

At the time of writing, it's still necessary to install the files in */boot* separately. It's possible that this requirement will go away in the future. There are two steps: first you build and install the boot files in the */boot* directory, then you install them on your boot disk. Assuming your system disk is the SCSI disk */dev/da0*, you would perform some of the following steps.

```
# cd /usr/src/sys                 build directory
# make install                    build and install the bootstraps
# bsdlabel -B da0                 Either, for a dedicated disk
# bsdlabel -B da0s1               Or, for a PC disk slice
# boot0cfg -B da0                 Or, booteasy for a dedicated PC disk
```

If you have a dedicated disk, which is normal on a non-Intel platform, use the first *bsdlabel* invocation to install the bootstrap (*boot1*) at the beginning of the disk. Otherwise, install *boot1* at the beginning of your FreeBSD slice and use *boot0cfg* to install the *boot0* boot manager at the beginning of the disk.