OpenBSD F.A.Q

March 2012

Contents

Chapter 1

Introduction to OpenBSD

1.1 What is OpenBSD?

The OpenBSD project produces a freely available, multi-platform 4.4BSD-based UNIX-like operating system. Our goals place emphasis on correctness, security, standardization, and portability.

This FAQ specifically covers only the most recent release of OpenBSD, version 5.0.

1.2 On what systems does OpenBSD run?

OpenBSD 5.0 runs on the following platforms:

- alpha - Download only
- amd64 - **Available on CD**
- armish - Download only
- hp300 - Download only
- hppa - Download only
- i386 - **Available on CD**
- landisk - Download only
- loongson - Download only
- macppc - **Available on CD**
- mvme68k - Download only
- mvme88k - Download only
- sgi - Download only
- socppc - Download only
- sparc - Download only
- sparc64 - **Available on CD**
- vax - Download only
- zaurus - Download only

Available on CD means the official CD set includes that platform and a number of packages. Base system CD ISO images can also be downloaded for most other platforms. These images are not the same as the official CD sets.

More information on OpenBSD platforms can be found on the Platforms page.

People sometimes ask why we support so many "odd" machines. The short answer is, "because we want to". If enough skilled people (sometimes, "enough" is only one really skilled person!) wish to maintain support for a platform, it will be supported. There are practical benefits to keeping OpenBSD multi-platform: when new platforms come out, the code tree is relatively free of portability-breaking bugs and design flaws. The OpenBSD platforms include 32 bit and 64 bit processors, little and big endian machines, and many different designs. And yes, supporting "unusual" platforms has helped produced a higher-quality code base for more "common" platforms.

1.3 Why might I want to use OpenBSD?

New users frequently want to know whether OpenBSD is superior to some other free UNIX-like operating system. That question is largely unanswerable and is the subject of countless (and useless) religious debates. Do not, under any circumstances, ask such a question on an OpenBSD mailing list.

Below are some reasons why we think OpenBSD is a useful operating system:

- OpenBSD runs on many different hardware platforms.
- OpenBSD is thought of by many security professionals as the most secure UNIX-like operating system, as the result of a never-ending comprehensive source code security audit.
- OpenBSD is a full-featured UNIX-like operating system available in source form at no charge.
- OpenBSD integrates cutting-edge security technology suitable for building firewalls and private network services in a distributed environment.
- OpenBSD benefits from strong ongoing development in many areas, offering opportunities to work with emerging technologies with an international community of programmers and end-users.
- OpenBSD attempts to minimize the need for customization and tweaking. For the vast majority of users, OpenBSD "Just Works" on their hardware for their application. Not only is tweaking and customizing rarely needed, it is actively discouraged.

However, whether OpenBSD is right for you is a question that only you can answer

1.4 Is OpenBSD really free?

OpenBSD is all free. The binaries are free. The source is free. All parts of OpenBSD have reasonable copyright terms permitting free redistribution. This includes the ability to REUSE most parts of the OpenBSD source tree, either

for personal or commercial purposes. OpenBSD includes NO further restrictions other than those implied by the original BSD license. Software which is written under stricter licenses cannot be included in the regular distribution of OpenBSD. This is intended to safeguard the free use of OpenBSD. For example, OpenBSD can be freely used for personal use, for academic use, by government institutions, by non-profit making organizations and by commercial organizations. OpenBSD, or parts of it, can also be freely incorporated into commercial products.

People sometimes ask if it bothers us that our free work is put into commercial products. The answer is, we would prefer that our good code be widely used rather than have commercial software vendors reimplement and create badly coded or incompatible alternative solutions to already solved problems. For example, it is likely that SSH is a widely used protocol due to this freedom, much more widely used than if restrictions had been placed on how people used the OpenSSH code. If a free SSH solution was not available for vendors to use (in their multitude of rapidly developed products), they would have written or purchased some crummy off-the shelf version instead.

This isn't to say we would object to financial or hardware support in thanks. In fact, it is stunning how little support of any kind comes from companies that depend upon OpenBSD (or OpenSSH) for their products, but there is no requirement of compensation.

For further reading on other popular licenses read: OpenBSD Copyright Policy.

The maintainers of OpenBSD support the project largely from their own pockets. This includes the time spent programming for the project, equipment used to support the many ports, the network resources used to distribute OpenBSD to you, and the time spent answering questions and investigating users' bug reports. The OpenBSD developers are not independently wealthy and even small contributions of time, equipment, and resources make a big difference.

1.5 How can I help support OpenBSD?

We are greatly indebted to the people and organizations that have contributed to the OpenBSD project. They are acknowledged by name on the donations page.

OpenBSD has a constant need for several types of support from the user community. If you find OpenBSD useful, you are strongly encouraged to find a way to contribute. If none of the suggestions below are right for you, feel free to propose an alternative by sending e-mail to `donations@openbsd.org`.

- Buy an OpenBSD CD set. It includes the current full release of OpenBSD, and is bootable on many platforms. It also generates revenue to support the OpenBSD project, and reduces the strain on network resources used to deliver the distribution via the Internet. This inexpensive three-CD set includes full source. Remember, your friends need their own copy!

- Donate money. The project has a constant need for cash to pay for equipment, network connectivity, and expenses relating to CD publishing. Manufacturing CDs requires an up-front out-of-pocket investment for the OpenBSD developers, without guaranteed return. Send e-mail to `donations@openbsd.org` to find out how to contribute. Even small donations make a profound difference.
- Donate equipment and parts. The project has a constant need for general and specific hardware. Items such as IDE, SCSI, SATA and SAS disks, and various types of RAM are always welcome. For other types of hardware such as computer systems and motherboards, you should inquire as to current need. Write to `donations@openbsd.org` to arrange for shipment.
- Donate your time and skills. Programmers who enjoy writing operating systems are naturally always welcome, but there are literally dozens of other ways that people can be useful. Follow mailing lists and help answer new-user questions.
- Help maintain documentation by submitting new FAQ material (to `faq@openbsd.org`). Form a local user group and get your friends hooked on OpenBSD. Make a case to your employer for using OpenBSD at work. If you're a student, talk to your professors about using OpenBSD as a learning tool for Computer Science or Engineering courses. It's also worth mentioning one of the most important ways you should not try to "help" the OpenBSD project: do not waste your time engaging in operating system flame wars. It does not help the project to find new users and can cause substantial harm to important relationships that developers have with other developers.

1.6 Who maintains OpenBSD?

OpenBSD is maintained by a development team spread across many different countries. The project is coordinated by Theo de Raadt, located in Canada.

1.7 When is the next release of OpenBSD?

The OpenBSD team makes a new release every six months, with target release dates in May and November. More information on the development cycle can be found here.

1.8 What is included with OpenBSD?

OpenBSD is distributed with a number of third-party software products, including:

- X.org 7.6, the X Window environment, with local patches. Installed with the x*.tgz install file sets.

- GCC versions 2.95.3, 3.3.5. and 4.2.1 (depending on your platform) GNU C Compiler. The OpenBSD team has added the Propolice stack protection technology, enabled by default, and used throughout the OpenBSD userland and by default on applications compiled on OpenBSD. Installed as part of the comp50.tgz file set.
- Perl 5.12.2, with patches and improvements from the OpenBSD team.
- Our improved and secured version of the Apache 1.3 web server. The OpenBSD team has added default chrooting, privilege revocation, and other security-related improvements. Also includes mod_ssl and DSO support.
- OpenSSL 1.0.0a, with patches and improvements from the OpenBSD team.
- Sendmail 8.14.5 mail server, with libmilter.
- BIND 9.4.2-P2 (plus patches) DNS server. OpenBSD has implemented many improvements in chroot operation and other security-related issues.
- Lynx 2.8.7rel.2 text web browser. With HTTPS and IPv6 support added, plus patches from the OpenBSD team.
- Sudo v1.7.2p8, allowing users to run individual commands as root.
- Ncurses 5.7
- KAME IPv6
- Heimdal 0.7.2 with patches
- Arla 0.35.7
- Binutils 2.15 with patches
- gdb 6.3 with patches
- OpenSSH 5.9
- OpenNTPD Secure and simple Network Time Protocol implementation
- OpenBGPD and OpenOSPFD routing applications

As can be seen, the OpenBSD team often patches third-party products (typically) to improve the security or quality of the code. In some cases, the user will see no difference in operation, in other cases, there ARE operational differences which may impact some users. Keep these enhancements in mind before blindly adding different versions of the same software. You may get a bigger version number, but a less secure system.

Of course, additional applications can be added through the OpenBSD packages and ports system.

1.9 What is new in OpenBSD 5.0?

The complete list of changes made to OpenBSD 4.9 to create OpenBSD 5.0 can be found on plus50.html, and highlights on the OpenBSD 5.0 Information page, however here are a few changes the OpenBSD team anticipate will require or warrant some special note to people upgrading or installing OpenBSD 5.0 who are familiar with older versions:

- **`/etc/rc.d` scripts used to start and manage system daemons** Previously used just for packages, now all system daemons start with scripts here.
- **Return of mac68k platform**
- **bce(4) enabled by default.** Hardware was limited to supporting 1GB or less RAM, but this problem is worked around in the OS now.
- **Base system and Xenocara manuals are now installed as source code** This makes grep(1) more useful in `/usr/share/man/` and `/usr/X11R6/man/` If both formatted and source versions of a manual page are installed, man(1) automatically displays the newer version of each page.
- **Big-memory support** Big-memory support is enabled on all possible architectures, including sparc64 and amd64.

1.10 Can I use OpenBSD as a desktop system?

This question is often asked in exactly this manner – with no explanation of what the asker means by "desktop". The only person who can answer that question is you, as it depends on what your needs and expectations are.

While OpenBSD has a great reputation as a "server" operating system, it can be and is used on the desktop. Many "desktop" applications are available through packages and ports. As with all operating system decisions, the question is: can it do the job you desire in the way you wish? You must answer this question for yourself.

It might be worth noting that a large amount of OpenBSD development is done on laptops.

1.11 Why is/isn't ProductX included?

People often ask why a particular product is or isn't included with OpenBSD. The answer is based on two things: the wishes of the developers and compatibility with the goals of the project. A product will not be included simply because it is "nifty" – it must also be "free" for use, distribution and modification by our standards. A product must also be stable and secure – a bigger version number does not always mean a better product.

License is often the biggest problem: we want OpenBSD to remain usable by any person anywhere in the world for any purpose.

Another major consideration is the wishes of the developers. The OpenBSD developers are the ultimate judges of what does and doesn't go into the project. Just because an application is "good" doesn't mean the OpenBSD project wishes to devote the resources needed to maintaining it, or that they will share other's enthusiasm about its place in OpenBSD.

Some commonly asked questions about third-party products:

- *Why is Sendmail included, it is "known insecure"?!*

  Sendmail has had an imperfect security record, however the Sendmail authors and maintainers have been very receptive to reworking their code to make it much more secure (and this is a sadly uncommon response). The recent security history of Sendmail is not much different than some of the supposedly "more secure" alternatives.

- *Why isn't Postfix included?*

  The license is not free, and thus can not be considered.

- *Why isn't qmail or djbdns included?*

  Neither program is what many Unix users "expect" out of a mail or DNS application.

- *Why is Apache included? It isn't needed by many people!*

  Because the developers want it.

- *Why isn't a newer version of Apache included?*

  The license on newer versions is unacceptable.

- *Why isn't bzip2 included instead of gzip?*

  Performance is horrible, and benefit is minimal. Impact on slower platforms, such as hp300 or VAX would be unacceptable.

- *Why isn't there a graphical or curses(3) based installer?*

  For a number of reasons, including the goal of keeping the installation boot media able to be a single floppy disk, the fact that one installer can be used on all platforms in all configurations, and the fact that after the second or third OpenBSD install, most users find the OpenBSD installation system among the fastest and easiest installers of any OS. Most developers and users greatly prefer the speed, power, and ease of use of the current installer to any of the more "colorful" or "pretty" installers on some other platforms.

- *Will ZFS be added to OpenBSD?*

  Not unless someone can convince Oracle to change the license for it to something compatible with OpenBSD policy. In most cases, these topics

have been discussed in painful detail on the mail lists, please see archives if you need more information.

Of course, If you wish to use one of these packages and your use is compatible with the license of the products, no one will stop you (that wouldn't be very free if we tried, would it?). However, your needs may change – you may not want to develop a "Killer Application" that you can't sell, distribute, or get rich from because you incorporated non-free software into it.

Chapter 2

Getting to know OpenBSD

2.1 Web Pages of Interest

The official website for the OpenBSD project is located at: `http://www.OpenBSD.org`.

A lot of valuable information can be found here regarding all aspects of the OpenBSD project.

The OpenBSD Journal is an OpenBSD-focused news and opinion site.

OpenBSDsupport.org is a site collecting "user maintained" documentation of varying quality, but often covering topics not in this FAQ or other official documentation.

Many users have set up sites and pages with OpenBSD specific information. As with everything on the Internet, a good search engine is going to make your life easier, as will a healthy dose of skepticism. As always, do not blindly enter commands you do not understand into your computer.

2.2 Mailing Lists

The OpenBSD project maintains several popular mailing lists which users should subscribe to and follow. To subscribe to a mailing list, send an e-mail message to `majordomo@openbsd.org`. That address is an automated subscription service. In the body of your message, on a single line, you should include a subscribe command for the list you wish to join. For example:

```
subscribe announce
```

The list processor will reply to you, asking for confirmation of your intent to join the list, so that others can not subscribe you to a flood of unwanted e-mail. The message will include instructions for several different ways to confirm, including a list server web page link, responding to the confirmation message or responding to majordomo@openbsd.org. Use whatever method is convenient to you. You will note that all three techniques involve a unique and time limited identifying number, such as `A56D-70D4-52C3`, again to make sure you are really the person who requested this mail list subscription (this is real "opt-in").

Once you have confirmed your intent to join, you will be immediately added to the list, and the list processor will notify you that you were successfully added.

To unsubscribe from a list, you will again send an e-mail message to `majordomo@openbsd.org`. It might look like this:

```
unsubscribe announce
```

If you have any difficulties with the mailing list system, please first read the help file which can be obtained by sending an e-mail message to `majordomo@openbsd.org` with a message body of "`help`".

Your subscription to the OpenBSD mail lists can also be maintained through the web interface at `http://lists.openbsd.org`

Some of the more popular OpenBSD mailing lists are:

- **announce** - Important announcements. This is a low-volume list.
- **security-announce** - Announcements of security issues. This is a low volume list.
- **misc** - General user questions and answers. This is the most active list, and should be the "default" for most questions.
- **bugs** - Bugs received via sendbug(1) and discussions about them.
- **source-changes** - Automated mailing of CVS source tree changes. Every time a developer commits a change to the OpenBSD source tree, CVS will send out a copy of the (usually brief) commit message via this list.
- **ports** - Discussion of the OpenBSD Ports Tree.
- **ports-changes** - Automated mailing of ports-specific CVS source tree changes.
- **advocacy** - Discussion on advocating OpenBSD, and topics that are just too off-topic for misc.

Before posting a question on misc or any other mailing list, please check the archives, for most common questions have been asked repeatedly. While it might be the first time you have encountered the problem or question, others on the mailing lists may have seen the same question several times in the last week, and may not appreciate seeing it again. If asking a question possibly related to hardware, *always include a* `dmesg(8)`*!*

You can find several archives, other mailing list guidelines and more information on the mailing lists page.

An unofficial mailing list that may be of interest to new users of OpenBSD and Unix is the OpenBSD Newbies list.

2.3 Manual Pages

OpenBSD comes with extensive documentation in the form of manual pages. Considerable effort is made to make sure the man pages are up-to-date and accurate. In all cases, the man pages are considered the authoritative source of information for OpenBSD.

To access the manual pages, be sure that you installed the `man50.tgz` file set.

Here is a list of some of the most useful manual pages for new users:

Getting Started

- afterboot(8) - things to check after the first complete boot.
- help(1) - help for new users and administrators.
- hier(7) - layout of filesystems.

- man(1) - display the on-line manual pages.
- intro(1) - introduction to general commands, also see the intros to the other sections of the manual: intro(2), intro(3), intro(4) (note: intro(4) is platform specific), intro(5), intro(6), intro(7), intro(8), and intro(9).
- adduser(8) - command for adding new users.
- vipw(8) - edit the master password file.
- disklabel(8) - read and write disk pack label.
- reboot, halt(8) - stop and restart the system.
- shutdown(8) - close down the system at a given time.
- dmesg(8) - redisplay the kernel boot messages
- sudo(8) - don't log in as root, but run commands as root.
- mg(1) - emacs-like text editor.

For most advanced users

- boot(8) - system bootstrapping procedures.
- boot_config(8) - how to change kernel configuration at boot.
- gcc-local(1) - OpenBSD-specific modifications to gcc(1)
- ifconfig(8) - configure network interface parameters.
- login.conf(5) - format of the login class configuration file.
- netstat(1) - show network status.
- release(8) - build an OpenBSD release.
- sendbug(1) - send a problem report (PR) about OpenBSD to a central support site.
- style(9) - OpenBSD kernel source code style guide.
- sysctl(8) - get or set kernel state.

You can find all the OpenBSD man pages on the web at http://www.openbsd.org/cgi-bin/man.cgi as well as on your computer if you install the `man50.tgz` file set.

In general, if you know the name of a command or a manual page, you can read it by executing "`man command`". For example: "`man vi`" to read about the vi editor. If you don't know the name of the command, or if "`man command`" doesn't find the manual page, you can search the manual page database by executing "`apropos something`" or "`man -k something`", where "something" is a likely word that might appear in the title of the manual page you're looking for. For example:

```
# apropos "time zone"
tzfile (5) - time zone information
zdump (8) - time zone dumper
zic (8) - time zone compiler
```

The parenthetical numbers indicate the section of the manual in which that page can be found. In some cases, you may find manual pages with identical names living in separate sections of the manual. For example, assume that you want to know the format of the configuration files for the cron daemon. Once you know the section of the manual for the page you want, you would execute "man n command", where n is the manual section number.

```
# man -k cron
cron (8) - clock daemon
crontab (1) - maintain crontab files for individual users
crontab (5) - tables for driving cron
# man 5 crontab
```

For many, having a hardcopy of the man page can be useful. Here are the guidelines to making a printable copy of a man page.

How do I display a man page source file (i.e. one whose filename ends in a number, like `tcpdump.8`)?

These are found throughout the src tree. The man pages are found in the tree unformatted, and many times, through the use of CVS, they will be updated. To view these pages, simply:

```
# mandoc ⟨file⟩ | more
```

How do I get a plain man page with no formatting or control characters?

This is helpful to get the man page straight, with no non-printable characters. Example:

```
# man ⟨command⟩ | col -b
```

How can I get a PostScript copy of a man page that's print-ready?

Note that ⟨file⟩ must be the man page source file (probably a file that ends in a number e.g. `tcpdump.8`). The PostScript versions of the man pages look very nice. They can be printed or viewed on-screen with a program like gv

(GhostView). GhostView can be found in our packages collection. Use the following `mandoc(1)` command options for getting a PostScript version from an OpenBSD system man page:

```
# mandoc -Tps ⟨file⟩ > outfile.ps
```

How do I generate compressed copies of the man pages?

For people who build their system from source, there are a number of options relating to the way in which man pages are built. These options can be placed in `/etc/mk.conf` (it may be necessary to create this file) and are included during system builds. One especially useful option is to generate compressed man pages in order to save disk space. These can be viewed in the normal way, using the man command. In order to set this, add the following to `/etc/mk.conf`:

```
MANZ=yes
```

Another useful option is to have the system build generate man pages in PostScript format, as well as ASCII text. This is done by setting the option `MANPS=yes` in `/etc/mk.conf`. See `mk.conf(5)` for further details.

What are info files?

Some of the documentation for OpenBSD comes in the form of info files, typically contained in `/usr/share/info`. This is an alternative form of documentation provided by GNU. Many of these files are more up to date than the manual pages provided by GNU, and can be accessed with the info(1) command. For example, to view information about the GNU compiler, `gcc(1)`, type:

```
# info gcc
```

After using info, you will really appreciate our man pages!

How do I get color man pages on XTerm?

The default configuration file for `xterm(1)` does not display color man pages. In order to get color output, copy the file `/etc/X11/app-defaults/XTerm-color` to your home directory, and rename it "`.Xdefaults`". Be careful not to overwrite any current settings in "`.Xdefaults`". This file contains all the settings you need to enable color in XTerm. However, three lines need to be uncommented before this can work:

```
!*VT100*colorULMode: on
!*VT100*underLine: off
!*VT100*colorBDMode: on
```

The rest of this file allows you to choose colors for various settings. The relevant ones to the man pages are:

```
*VT100*colorUL: yellow
*VT100*colorBD: white
```

That produces rather hellish looking man pages, so customize as necessary: may we suggest red for "`colorUL`" and magenta for "`colorBD`"? There is also a man page viewer for X11 available, `xman(1)`, which provides an alternative (graphical) interface to the manual pages. See the manual pages for xterm and xman for more information.

How do I write my own manual page?

If you wish to write your own man page for an application you have written, there is a handy reference guide provided in `mdoc(7)`.

2.4 Reporting Bugs

Before crying "Bug!", please make sure that is really what you are dealing with. If instead, you are not understanding how something is done in OpenBSD or how it works, and can't find out how to resolve the problem using the manual pages or the OpenBSD website, use the mail lists (usually `misc@openbsd.org`) to request help. If this is your first OpenBSD experience, be realistic: you probably did not discover an unknown bug. Also note that faulty hardware can mimic a software bug, please verify the current condition of your hardware before deciding you have found a "bug".

Finally, before submitting any bug report, please read `http://www.openbsd.org/report.html`.

Proper bug reporting is one of the most important responsibilities of end users. Very detailed information is required to diagnose most serious bugs. Developers frequently get bugs reports via e-mail such as this:

```
From: joeuser@example.com
To: bugs@openbsd.org
Subject: HELP!!!

I have a PC and it won't boot!!!!! It's a 486!!!!!
```

Hopefully most people understand why such reports get summarily deleted. All bug reports should contain detailed information. If Joe User had really expected someone to help find this bug, he or she would have supplied more information... something like this:

```
From: smartuser@example.com
To: bugs@openbsd.org
Subject: 3.3-beta panics on a SPARCStation2
```

```
OpenBSD 3.2 installed from an official CD-ROM installed and ran fine
on this machine.

After doing a clean install of 3.3-beta from an FTP mirror, I find the
system randomly panics after a period of use, and predictably and
quickly when starting X.

This is the dmesg output:

OpenBSD 3.3-beta (GENERIC) #9: Mon Mar 17 12:37:18 MST 2003
    deraadt@sparc.openbsd.org:/usr/src/sys/arch/sparc/compile/GENERIC
real mem = 67002368
avail mem = 59125760
using 200 buffers containing 3346432 bytes of memory
bootpath: /sbus@1,f8000000/esp@0,800000/sd@1,0
mainbus0 (root): SUNW,Sun 4/75
cpu0 at mainbus0: CY7C601 @ 40 MHz, TMS390C602A FPU; cache chip bug
- trap page uncached
cpu0: 64K byte write-through, 32 bytes/line, hw flush cache enabled
memreg0 at mainbus0 ioaddr 0xf4000000
clock0 at mainbus0 ioaddr 0xf2000000: mk48t02 (eeprom)
timer0 at mainbus0 ioaddr 0xf3000000 delay constant 17
auxreg0 at mainbus0 ioaddr 0xf7400003
zs0 at mainbus0 ioaddr 0xf1000000 pri 12, softpri 6
zstty0 at zs0 channel 0 (console i/o)
zstty1 at zs0 channel 1
zs1 at mainbus0 ioaddr 0xf0000000 pri 12, softpri 6
zskbd0 at zs1 channel 0: reset timeout
zskbd0: no keyboard
zstty2 at zs1 channel 1: mouse
audioamd0 at mainbus0 ioaddr 0xf7201000 pri 13, softpri 4
audio0 at audioamd0
sbus0 at mainbus0 ioaddr 0xf8000000: clock = 20 MHz
dma0 at sbus0 slot 0 offset 0x400000: rev 1+
esp0 at sbus0 slot 0 offset 0x800000 pri 3: ESP100A, 25MHz, SCSI ID 7
scsibus0 at esp0: 8 targets
sd0 at scsibus0 targ 1 lun 0: <SEAGATE, ST1480 SUN0424, 8628> SCSI2 0/direct fixed
sd0: 411MB, 1476 cyl, 9 head, 63 sec, 512 bytes/sec, 843284 sec total
sd1 at scsibus0 targ 3 lun 0: <COMPAQPC, DCAS-32160, S65A> SCSI2 0/direct fixed
sd1: 2006MB, 8188 cyl, 3 head, 167 sec, 512 bytes/sec, 4110000 sec total
le0 at sbus0 slot 0 offset 0xc00000 pri 5: address 08:00:20:13:10:b9
le0: 16 receive buffers, 4 transmit buffers
cgsix0 at sbus0 slot 1 offset 0x0: SUNW,501-2325, 1152x900, rev 11
wsdisplay0 at cgsix0
wsdisplay0: screen 0 added (std, sun emulation)
```

```
fdc0 at mainbus0 ioaddr 0xf7200000 pri 11, softpri 4: chip 82072
fd0 at fdc0 drive 0: 1.44MB 80 cyl, 2 head, 18 sec
root on sd0a
rootdev=0x700 rrootdev=0x1100 rawdev=0x1102


This is the panic I got when attempting to start X:

panic: pool_get(mclpl): free list modified: magic=78746572; page 0xfaa93000;
 item addr 0xfaa93000
Stopped at      Debugger+0x4:   jmpl            [%o7 + 0x8], %g0
RUN AT LEAST 'trace' AND 'ps' AND INCLUDE OUTPUT WHEN REPORTING THIS PANIC!
DO NOT EVEN BOTHER REPORTING THIS WITHOUT INCLUDING THAT INFORMATION!
ddb> trace
pool_get(0xfaa93000, 0x22, 0x0, 0x1000, 0x102, 0x0) at pool_get+0x2c0
sosend(0x16, 0xf828d800, 0x0, 0xf83b0900, 0x0, 0x0) at sosend+0x608
soo_write(0xfac0bf50, 0xfac0bf70, 0xfac9be28, 0xfab93190, 0xf8078f24, 0x0)
at soo_write+0x18
dofilewritev(0x0, 0xc, 0xfac0bf50, 0xf7fff198, 0x1, 0xfac0bf70) at
dofilewritev+0x12c
sys_writev(0xfac87508, 0xfac9bf28, 0xfac9bf20, 0xf80765c8, 0x1000, 0xfac0bf70)
at sys_writev+0x50
syscall(0x79, 0xfac9bfb0, 0x0, 0x154, 0xfcffffff, 0xf829dea0) at syscall+0x220
slowtrap(0xc, 0xf7fff198, 0x1, 0x154, 0x1, 0xfac87508) at slowtrap+0x1d8
ddb> ps
   PID   PPID   PGRP    UID  S       FLAGS  WAIT       COMMAND
 27765   8819  29550      0  3        0x86  netio      xconsole
  1668  29550  29550      0  3      0x4086  poll       fvwm
 15447  29550  29550      0  3     0x44186  poll       xterm
  8819  29550  29550     35  3      0x4186  poll       xconsole
  1238  29550  29550      0  3      0x4086  poll       xclock
 29550  25616  29550      0  3      0x4086  pause      sh
  1024  25523  25523      0  3     0x40184  netio      XFree86
*25523  25616  25523     35  2     0x44104             XFree86
 25616  30876  30876      0  3      0x4086  wait       xinit
 30876  16977  30876      0  3      0x4086  pause      sh
 16977      1  16977      0  3      0x4086  ttyin      csh
  5360      1   5360      0  3        0x84  select     cron
 14701      1  14701      0  3     0x40184  select     sendmail
 12617      1  12617      0  3        0x84  select     sshd
 27515      1  27515      0  3       0x184  select     inetd
  1904      1   1904      0  2        0x84             syslogd
  9125      1   9125      0  3        0x84  poll       dhclient
     7      0      0      0  3    0x100204  crypto_wa  crypto
     6      0      0      0  3    0x100204  aiodoned   aiodoned
     5      0      0      0  3    0x100204  syncer     update
```

```
    4      0      0      0  3    0x100204  cleaner    cleaner
    3      0      0      0  3    0x100204  reaper     reaper
    2      0      0      0  3    0x100204  pgdaemon   pagedaemon
    1      0      1      0  3      0x4084  wait       init
    0     -1      0      0  3     0x80204  scheduler  swapper

Thank you!
```

See `report.html` for more information on creating and submitting bug reports. Detailed information about your hardware is necessary if you think the bug *could be in any way* related to your hardware or hardware configuration. Usually, `dmesg(8)` output is sufficient in this respect. A detailed description of your problem is necessary. You will note that the dmesg described the hardware, the text explained why Smart User thought the system was not broken (ran 3.2 properly), how this crash was caused (starting X), and the output of the debugger's "`ps`" and "`trace`" commands. In this case, Smart User provided output captured on a serial console; if you can not do that, you will have to use paper and pencil to record the crash. (This was a real problem, and the information in the above report helped lead to a repair of this issue which impacted Sun4c systems.)

If Smart User had a working OpenBSD system from which he wanted to submit a bug report, he would have used the `sendbug(1)` utility to submit his bug report to the GNATS problem tracking system. Obviously you can't use `sendbug(1)` when your system won't boot, but you should use it whenever possible. You will still need to include detailed information about what happened, the exact configuration of your system, and how to reproduce the problem. The `sendbug(1)` command requires that your system be able to send electronic mail successfully on the Internet. Note that the mail server uses `spamd(8)` based greylisting, so it may take half an hour or so before the mail server accepts your bug report, so please be patient.

After submitting a bug report via `sendbug(1)`, you may be contacted by developers for additional information or with patches that need testing. You can also monitor the archives of the `bugs@openbsd.org` mailing list, details on the mailing list page.

More on getting useful info for developers

Here are a few additional tips:

Lost the "Panic message"?

Under some circumstances, you may lose the very first message of a panic, stating the reason for the panic. This is a very important message, so you want to report it, as well. You can get this back by using the "`show panic`" command in ddb> like this:

```
ddb> show panic
0:      kernel: page fault trap, code=0
ddb>
```

In this case, the panic string was "`Kernel:  page fault trap, code=0`"

Special note for SMP systems:

You should get a "`trace`" from each processor as part of your report:

```
ddb0> trace
pool_get(d05e7c20,0,dab19ef8,d0169414,80) at pool_get+0x226
fxp_add_rfabuf(d0a62000,d3c12b00,dab19f10,dab19f10) at fxp_add_rfabuf+0xa5
fxp_intr(d0a62000) at fxp_intr+0x1e7
Xintr_ioapic0() at Xintr_ioapic0+0x6d
--- interrupt ---
idle_loop+0x21:
ddb0> machine ddbcpu 1
Stopped at      Debugger+0x4:   leave
ddb1> trace
Debugger(d0319e28,d05ff5a0,dab1bee8,d031cc6e,d0a61800) at Debugger+0x4
i386_ipi_db(d0a61800,d05ff5a0,dab1bef8,d01eb997) at i386_ipi_db+0xb
i386_ipi_handler(b0,d05f0058,dab10010,d01d0010,dab10010) at i386_ipi_handler+0x
4a
Xintripi() at Xintripi+0x47
--- interrupt ---
i386_softintlock(0,58,dab10010,dab10010,d01e0010) at i386_softintlock+0x37
Xintrltimer() at Xintrltimer+0x47
--- interrupt ---
idle_loop+0x21:
ddb1>
```

Repeat the "`machine ddbcpu x`" followed by "`trace`" for each processor in your machine.

How to gather further information from a kernel crash.

A typical kernel crash on OpenBSD might look like this: (things to watch for are marked with bold font)

```
kernel: page fault trap, code=0
Stopped at    _pf_route+0x263:        mov     0x40(%edi)%edx
ddb>
```

The first command to run from the `ddb>` prompt is "`trace`" (see `ddb(4)` for details):

```
ddb> trace
_pf_route(e28cb7e4,e28bc978,2,1fad,d0b8b120) at _pf_route+0x263
```

```
_pf_test(2,1f4ad,e28cb7e4,b4c1) at _pf_test+0x706
_pf_route(e28cbb00,e28bc978,2,d0a65440,d0b8b120) at _pf_route+0x207
_pf_test(2,d0a65440,e28cbb00,d023c282) at _pf_test+0x706
_ip_output(d0b6a200,0,0,0,0) at _ip_output+0xb67
_icmp_send(d0b6a200,0,1,a012) at _icmp_send+0x57
_icmp_reflect(d0b6a200,0,1,0,3) at _icmp_reflect+0x26b
_icmp_input(d0b6a200,14,0,0,d0b6a200) at _icmp_input+0x42c
_ipv4_input(d0b6a200,e289f140,d0a489e0,e289f140) at _ipv4_input+0x6eb
_ipintr(10,10,e289f140,e289f140,e28cbd38) at _ipintr+0x8d
Bad frame pointer: 0xe28cbcac
ddb>
```

This tells us what function calls lead to the crash.

To find out the particular line of C code that caused the crash, you can do the following: Find the source file where the crashing function is defined in. In this example, that would be `pf_route()` in `sys/net/pf.c`. Recompile that source file with debug information:

```
# cd /usr/src/sys/arch/$(uname -m)/compile/GENERIC/
# rm pf.o
# DEBUG=-g make pf.o
```

Then use `objdump(1)` to get the disassembly:

```
# objdump --line --disassemble --reloc pf.o >pf.dis
```

In the output, grep for the function name (pf_route in our example):

```
# grep "<_pf_route>:" pf.dis
00007d88 <_pf_route>:
```

Take this first hex number and add the offset from the 'Stopped at' line: **0x7d88** + **0x263** == **0x7feb**. Scroll down to that line (the assembler instruction should match the one quoted in the 'Stopped at' line), then up to the nearest C line number:

```
# more pf.dis
/usr/src/sys/arch/i386/compile/GENERIC/../../../../net/pf.c:3872
    7fe7:       0f b7 43 02             movzwl 0x2(%ebx),%eax
    7feb:       8b 57 40                mov    0x40(%edi),%edx
    7fee:       39 d0                   cmp    %edx,%eax
    7ff0:       0f 87 92 00 00 00       ja     8088 <_pf_route+0x300>
```

So, it's precisely line **3872** of pf.c that crashes:

```
# cat -n pf.c | head -n 3872 | tail -n 1
3872          if ((u_int16_t)ip->ip_len <= ifp->if_mtu) {
```

Note that the kernel that produced the crash output and the object file for objdump must be compiled from the exact same source file, otherwise the offsets won't match.

If you provide both the `ddb>` trace output and the relevant objdump section, that's very helpful.

Chapter 3

Getting started with OpenBSD

3.1 Buying an OpenBSD CD set

Purchasing an OpenBSD CD set is generally the best way to get started. Visit the ordering page to purchase your copy: OpenBSD ordering page.

There are many good reasons to own an OpenBSD CD set:

- CD sales support ongoing development of OpenBSD.
- Development of a multi-platform operating system requires constant investment in equipment.
- Your support in the form of a CD set purchase has a real impact on future development.
- The CDs contains binaries (and source) for the most popular supported platforms.
- The CDs are bootable on several platforms, and can be used to bootstrap a machine without a pre-existing installed operating system.
- The CDs are useful for bootstrapping even if you choose to install a snapshot.
- Installing from CD is faster! Installing from CD preserves network connectivity resources.
- OpenBSD CDs always come with very nice stickers. Your system isn't fully complete without these. You can only get these stickers by buying a CD set or donating hardware.
- OpenBSD CD sets come with an assortment of useful and popular packages. The CD set is complete enough to bring up a full work and development environment without any network connection at all.

If you're installing a release version of OpenBSD, you should use a official CD set.

3.2 Buying OpenBSD T-Shirts

Yes, OpenBSD has T-shirts for your wearing enjoyment. You can view these at the OpenBSD T-shirts page. Enjoy :)

3.3 Does OpenBSD provide an ISO image for download?

For select platforms, yes!

Most platforms that are able to boot from CD-ROM have ISO images available for download which can be used to create bootable OpenBSD install CD-ROMs. Both "full install" (all files needed for a full OpenBSD install, named `install50.iso`) and "boot only" (for installing from a network file source, named `cd50.iso`) ISOs are provided for these platforms.

Note, these ISO files are not the same as the official CD set. These images are for single platforms, and do not include any of the pre-compiled packages, stickers, or artwork that the official CD set does.

However, ISO file installation is NOT the optimum installation method for many people. It is still usually faster and simpler to download the boot media and then install just the portions needed. However, for those who wish to do a number of installations, or can not figure out how to drop ten files on a CD-ROM or set up a local HTTP/FTP server, ISOs are available.

The OpenBSD project does not make the ISO images used to master the official CDs available for download. The reason is simply that we would like you to buy the CD sets to help fund ongoing OpenBSD development. The official OpenBSD CD-ROM layout is copyright Theo de Raadt. Theo does not permit people to redistribute images of the official OpenBSD CDs. As an incentive for people to buy the CD set, some extras are included in the package as well (artwork, stickers etc).

Note that only the CD layout is copyrighted, OpenBSD itself is free. Nothing precludes someone else from downloading OpenBSD and making their own CD.

For those that need a bootable CD for their system, bootdisk ISO images (named `cd50.iso`) are available for a number of platforms which will then permit the rest of the system to be installed via HTTP/FTP. These ISO images are only a few megabytes in size, and contain just the installation tools, not the actual file sets.

3.4 Downloading via HTTP, FTP or AFS

There are numerous international mirror sites offering HTTP and FTP access to OpenBSD releases and snapshots. AFS access is also available. You should always use the site nearest to you. Before you begin fetching a release or snapshot, you may wish to use `ping(8)` and `traceroute(8)` to determine which mirror site is nearest to you and whether that mirror is performing adequately. Of course, your OpenBSD release CD is always closer than any mirror. Access information is here:

OpenBSD Download page.

3.5 Selecting Hardware

Selecting appropriate hardware to run your OpenBSD system on is important, as it can mean the difference between success and failure of a project.

If you are shopping for a new PC, whether you are buying it piece by piece or completely pre-built, you want to make sure first that you are buying reliable parts. In the PC world, this is not easy. **Bad or otherwise unreliable or mismatched parts can make OpenBSD run poorly and crash often**. The best advice we can give is to be careful, and buy brands and parts that have been reviewed by an authority you trust. Sometimes, a higher-price machine is a better quality machine. Other times, it is simply more expensive.

There are certain things that will help bring out the maximum performance of your system:

- **Let the application chose the hardware**: It is usually better to make an adjustment to the hardware you were planning on using rather than compromising on your application design because you have something you "really wanted to use".
- **Identify your bottlenecks**: Don't pay extra for the cutting-edge processor if your application is restricted by disk I/O. Don't pay for fast disk if your system is restricted by network speed. Don't pay for much of anything if your bottleneck is a 128kbps DSL line.
- **Keep it simple**: Simple hardware usually has simple problems. Complex hardware that isn't supposed to ever break may take you a long time to repair when it breaks anyway.
- **Use hardware you understand, or learn the new hardware before you implement a production system**: Regardless of the technical merits of the hardware, committing to use a particular type of hardware before you have become familiar with it, both how it works *and how it fails*, is foolish.
- **Use multiple disks**: Instead of buying one large disk, buy multiple smaller disks. While this may cost more, distributing the load over multiple spindles will decrease the amount of time necessary to access data on the disks. And, with more spindles, you can get more reliability and faster data access with RAID.
- **Break up large blocks of storage**: Many people make the mistake of designing a system that stores large amounts of data to have one Big Block of Storage. This is usually a bad design. You will usually find it much better to break up your storage into manageable blocks. This has many advantages, two of the biggest being that you can add more storage later when you need it easily (you weren't going to get the requirements estimate right, no one ever does), and you can buy a small amount of storage now, and add much more later when the price will have most likely dropped and the capacity will have increased.
- **Avoid cheap network adapters**: OpenBSD supports a plethora of cheap network adapters. These adapters work great in home systems,

and low or moderate throughput business and research environments. But, if you need high throughput and low impact on your server, you are better off buying a quality network adapter. Unfortunately, some name-brand adapters are not much better than the cheap adapters, and some potentially good adapters do not have accurate documentation available to write good drivers. Gigabit adapters often perform better than 10Mbps/100Mbps adapters, even when used on slower speed networks, due to superior buffering.

3.6 What is an appropriate "first system" to learn OpenBSD on?

While OpenBSD will run on a smaller, older and less powerful computer than just about any other modern OS, if you are just getting started with OpenBSD, using too little machine can be frustrating. The following guidelines are ONLY guidelines, OpenBSD will run very well on much more modest equipment than is listed here, but it may be more frustrating than needed for a first-time user.

- **Platform**: Use a platform you are familiar with already. When you are learning a new operating system, it is a very bad time to also be learning a new platform. We'll assume you are going to be using the i386 platform here, as it is probably the one most people are familiar with.
- **Processor**: 100MHz Pentium or better processor. Yes, OpenBSD will run on a 25MHz 80486, but you won't want to do the experimenting and messing up and reloading you need to do to really get to know the system on a slow machine. The primary irritation you will first encounter with a processor slower than this is the time it takes to SSH into the box. If you wish to run X, you probably want to move up to at least 200MHz. X actually runs pretty well on a slower machine once loaded, but it takes a while to load and start.
- **64M RAM or better**: If you wish to run X, 128M would be a better starting point.
- **Hard disk**: A 1G hard disk will give you an easy install of a simple system, such as a firewall, DNS server, or similar. If you wish to rebuild the system from source, you will probably want a 4G disk, and if you wish to rebuild X as well, you will want 6G or bigger. IDE is recommended to start with. If you have a much larger disk, don't feel the obligation to allocate all the disk initially – there is nothing wrong with leaving 72G of an 80G hard disk unallocated if all you need is 8G.
- **Network adapter**: Use a PCI adapter. If you are planning on putting multiple network adapters in the machine, write the MAC address on the spine of the card before putting the cover on. While the urge to use

that old ISA adapter you have may be strong, resist. You probably don't remember how to properly configure it.

- **Multibooting**: For your first OpenBSD installation, don't attempt to multiboot with another OS. Multibooting is a difficult process to get right, and you must understand all the OSs involved well before attempting this, which is clearly not the case on your first installation. It is very possible you could accidently delete all data on the system. Rather, use a dedicated computer, or at least, a dedicated disk on a machine.

- **Laptops**: While many laptops work very well with OpenBSD, they are sometimes not the easiest systems to get running well, so a laptop might not be the best choice for your first OpenBSD install. However, once you are comfortable with OpenBSD, a laptop can be a very useful tool.

- **New hardware**: Brand new, cutting-edge hardware is sometimes not yet supported by OpenBSD, so for your first OpenBSD system, a slightly older machine is recommended.

Obviously, "more the better" to a point. Some popular applications, seemingly can use as much processor and memory as you can throw at the system.

Chapter 4

OpenBSD 4.9 Installation Guide

4.1 Overview of the OpenBSD installation procedure

OpenBSD has long been respected for its simple and straight forward installation process, which is consistent across all platforms.

All platforms use a very similar installation procedure, however there are some minor differences in details on a few platforms. In all cases, you are urged to read the platform-specific INSTALL document in the *platform* directory on the CD-ROM or FTP sites (for example, `i386/INSTALL.i386`, `macppc/INSTALL.macppc` or `sparc/INSTALL.sparc`).

The OpenBSD installer is a special kernel with a number of utilities and install scripts embedded in a pre-loaded RAM disk. After this kernel is booted, the operating system is extracted from a number of compressed `tar(1)` (`.tgz`) files from a source other than this pre-loaded RAM disk. There are several ways to boot this install kernel:

- **Floppy disk**: OpenBSD can be installed on many platforms by booting an installer from a single floppy disk. However, due to space constraints, some larger platforms (sparc64, amd64, alpha) do not have some utilities which may be important to you, such as a DHCP client to configure the network. For these platforms, you may do better with the CD install. However, for platforms like i386 and sparc, you will find the boot floppy very complete. Floppy disk images are provided which can be used to create an install floppy on another Unix-like system, or on a Windows system. Typical file names are `floppy50.fs`, though several platforms have multiple floppy images available.

- **CD-ROM**: On several platforms a CD-ROM image (`cd50.iso` for just booting, or `install50.iso` for the entire install) is provided allowing creation of a bootable CD-ROM. Existing partition: The RAM disk kernel can be booted off an already existing partition for an upgrade or reinstall.

- **Network**: Some platforms support booting over a network (for example using PXE or other network boot). Writing a file system image to disk (miniroot): a filesystem image that can be written to an existing partition, and then can be booted.

- **Bootable Tape**: Some platforms support booting from tape. These tapes can be made following the INSTALL.*platform* instructions.

Not every platform supports all boot options:

- alpha: Floppy, CD-ROM, network, writing a floppy image to hard disk.

- amd64: Floppy, CD-ROM, network.

- armish: Varies by machine.

- hp300: CD-ROM, network.
- hppa: Network.
- i386: Floppy, CD-ROM, network.
- landisk: miniroot, installed using another computer.
- macppc: CD-ROM, network.
- mvme68k: Network, bootable tape.
- mvme88k: Network, bootable tape.
- sparc: Floppy, CD-ROM, network, writing image to existing swap partition, bootable tape.
- sparc64: Floppy (U1/U2 only), CD-ROM, network, writing image to existing partition.
- vax: Floppy, network.
- zaurus: Boot bsd.rd from Linux partition. See INSTALL.zaurus for details.

All platforms can also use a `bsd.rd` to reinstall or upgrade.

Once the install kernel is booted, you have several options of where to get the install file sets. Again, not every platform supports every option.

- **CD-ROM**: Of course, we prefer you use the Official CD-ROM set, but you can also use install50.iso or you can also make your own.
- **FTP**: Either one of the OpenBSD FTP mirror sites or your own local FTP server holding the file sets.
- **HTTP**: Either one of the OpenBSD HTTP mirror sites or your own local web server holding the file sets. Local disk partition: In many cases, you can install file sets from another partition on a local hard disk. For example, on i386, you can install from a FAT partition or a CD-ROM formatted in ISO9660, Rock Ridge or Joliet format. In some cases, you will have to manually mount the file system before using it.
- **NFS**: Some platforms support using NFS mounts for the file sets.
- **Tape**: File sets can also be read from a supported tape. Details on creating the tape are in the INSTALL.platform document.

4.2 Pre-installation checklist

Before you start your install, you should have some idea what you want to end up with. You will want to know the following items, at least:

- Machine name
- Hardware installed and available

  Verify compatibility with your platform's hardware compatibility page

  If ISA, you also need to know hardware settings, and confirm they are as OpenBSD requires.
- Install method to be used (CD-ROM, FTP, etc.)
- Should an important bug be found, how will the system be patched?

  If done locally, you will need to have sufficient space available for the source tree and building it.

  Otherwise, you will need access to another machine to build a patched release on.
- Desired disk layout

  Does existing data need to be saved elsewhere?

  Will OpenBSD coexist on this system with another OS? If so, how both will be booted? Will you need to install a "boot manager"?

  Will the entire disk be used for OpenBSD, or do you want to keep an existing partition/OS (or space for a future one)?

  How do you wish to sub-partition the OpenBSD part of your disk?
- Network settings, if not using DHCP:

  Domain name

  Domain Name Server(s) (DNS) address

  IP addresses and subnet masks for each NIC

  Gateway address
- Will you be running the X Window System?

4.3 Creating bootable OpenBSD install media

As examples, we will look at the installation images available for the i386 and sparc platforms.

The i386 platform has six separate installation disk images to choose from:

- **cd50.iso** is an ISO9660 image that can be used to create a bootable CD with most popular CD-ROM creation software on most platforms. This image has the widest selection of drivers, and is usually the recommended choice if your hardware can boot from a CDROM.
- **cdemu50.iso** is an ISO9660 image, using "floppy emulation" booting, using a 2.88M floppy image. It is hoped that few people will need this image – most people will use `cd50.iso`, only use `cdemu50.iso` if `cd50.iso` doesn't work for you.
- **install50.iso** is an ISO9660 image, containing all the standard install files. This file can be used to create a CD that can do a stand-alone OpenBSD install.
- **floppy50.fs** (Desktop PC) supports many PCI and ISA NICs, IDE, SATA and simple SCSI adapters and some PCMCIA support. Most users will use this image if booting from a floppy
- **floppyB50.fs** (Servers) supports many RAID controllers, and some of the less common SCSI adapters. However, support for many standard SCSI adapters and many EISA and ISA NICS has been removed.
- **floppyC50.fs** (Laptops) supports the CardBus and PCMCIA devices found in many laptops.

The sparc platform has four separate installation disk images to choose from:

- **floppy50.fs**: Supports systems with a floppy disk.
- **cd50.iso** An ISO image usable to make your own CD for booting SPARC systems with a CD-ROM.
- **miniroot50.fs** Can be written to a swap partition and booted.
- **install50.iso** is an ISO9660 image, containing all the standard install files. This file can be used to create a CD that can do a stand-alone OpenBSD install.

On modern platforms, you are best advised to use the CDROM boot images, as in some of the "bigger" platforms (such as amd64, sparc64), the floppy images have had to have a lot of drivers and utilities cut out, which can make installation much more difficult. Older platforms, such as i386 and sparc, are still quite installable from floppy.

4.3.1 Making a CD-ROM

You can create a CD-ROM using the `cd50.iso` or `install50.iso` files. The exact details here are left to the reader to determine with the tools they have at their disposal.

In OpenBSD, you can create a CD from an ISO image using `cdio(1)`:

```
# cdio tao cd50.iso
```

Most CD recorders sold for Windows and Macintosh systems come with software that can burn ISO images to blank media. If yours does not, there are various no-cost applications that can do this for you.

Other Unix-like systems use applications such as cdrkit.

4.3.2 Creating floppies on Unix

Precise details and device names may vary from Unix variant to Unix variant; use what is appropriate for your system.

To create a formatted floppy, use the `fdformat(1)` command to both format and check for bad sectors.

```
# fdformat /dev/rfd0c
Format 1440K floppy '/dev/rfd0c'? (y/n): y
Processing VVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVV done.
```

If your output is like the above example, then the disk is OK. However, if you do not see ALL "V"'s then the disk is most likely bad, and you should try a new one.

To write the image file to the disk, use dd(1). An example usage of dd(1) is below:

```
# dd if=floppy50.fs of=/dev/rfd0c bs=32k
```

Once the image is written, check to make sure that the copied image is the same as the original with the cmp(1) command. If the diskette is identical to the image, you will just see another prompt.

```
# cmp /dev/rfd0c floppy50.fs
```

4.3.3 Creating floppies on Windows

To prepare a floppy in Windows, first use the native formatting tools to format the disk, verifying that the disk has no bad sectors.

To write the installation image to the prepared floppy you can use ntrw.exe, which can be downloaded from the tools directory on any of the FTP mirrors.

Example usage of ntrw:

```
C:\> ntrw floppy50.fs a:
3.5", 1.44MB, 512 bytes/sector
bufsize is 9216
1474560 bytes written
```

4.4 Booting OpenBSD install media

Booting i386/amd64

Booting an install media on the i386 and amd64 PC platforms is nothing new to most people. Your system will have to be instructed to boot from whatever media you have chosen to use, usually through a BIOS setup option. If you want to boot from CD, your system BIOS must be able to and be set to boot from CD. Some older systems do not have this option, and you must use a floppy for booting your installation image. Don't worry though; even if you boot from floppy you can still install from the CD if it is supported by OpenBSD (i.e., almost all IDE drives). You can also install by booting bsd.rd from an existing OpenBSD partition, or over the network using the PXE boot process.

Booting sparc/sparc64

NOTE: On the sparc64 platform, only the SBus machines (Ultra 1, Ultra 2) are bootable from floppy. You will need the system to be at a monitor ROM prompt, which typically looks like "`ok` ". If you are using a Sun keyboard, press and hold "STOP" while tapping "A". If using a serial console, a BREAK should return you to the monitor prompt.

Use the following command to boot from the floppy:

```
ok boot floppy
```

Usually, you can boot from the CDROM drive of a Sun system from the boot prompt by typing '`boot cdrom`':

```
ok boot cdrom
```

4.5 Performing a simple install

OpenBSD's new installer is designed to install and configure OpenBSD in a very usable default configuration with very little user intervention. In fact, you can often just hit *ENTER* a number of times to get a good OpenBSD install, moving your hands to the rest of the keyboard only to enter the root password. The installer will create a partitioning plan based on the size of your hard disk. While this will NOT be a perfect layout for all people, it provides a good starting point and a good overall strategy for figuring out what you need.

We will start with a very simple install, with brief discussions of the options provided, and using the magic of hypertext links, allow you to read more on the topics that interest you and explore your options.

Installation notes for each platform are on the install CDs and FTP servers, in the file `INSTALL.<plat>`, where ¡plat¿ is your platform, for instance, `i386`.

4.5.1 Starting the install

Whatever your means of booting is, it is now time to use it. During the boot process, the kernel and all of the programs used to install OpenBSD are loaded

into memory. Once the install kernel is booted, the boot media is no longer needed, everything runs from the RAM disk. You can actually remove the CD or floppy you booted from at this point, assuming you don't need the CD for installation files. At almost any point during the OpenBSD install process, you can terminate the current install attempt by hitting CTRL-C and can restart it without rebooting by running `install` at the shell prompt. You can also enter a "!" at most places in the installation to get to a shell prompt, then exit the shell to return to the installer.

When your boot is successful, you will see a lot of text messages scroll by. This text, on many architectures in white on blue, is the `dmesg`, the kernel telling you what devices have been found and how they are hooked to other devices. A copy of this text is saved as `/var/run/dmesg.boot`.

Then, you will see the following:

```
...
root on rd0a swap on rd0b dump on rd0b
erase ^?, werase ^W, kill ^U, intr ^C, status ^T

Welcome to the OpenBSD/i386 5.0 installation program.
(I)nstall, (U)pgrade or (S)hell? i
```

And with that, we reach our first question. You have the three options shown:

- **Install**: load OpenBSD onto the system, overwriting whatever may have been there. Note that it is possible to leave some partitions untouched in this process, such as a `/home`, but otherwise, assume everything else is overwritten.

- **Upgrade**: Install a new set of install files on this machine, but do not overwrite any configuration information, user data, or additional programs. No disk formatting is done, nor are the `/etc` or `/var` directories overwritten. A few important notes:

  You will not be given the option of installing the `etc50.tgz` file. After the install, you will have to manually merge the changes of `etc50.tgz` into your system before you can expect it to be fully functional. This is an important step which must be done, as otherwise certain key services (such as `pf(4)`) may not start.

  The Upgrade process is not designed to skip releases! While this will often work, it is not supported. For OpenBSD 5.0, upgrading 4.9 to 5.0 is the only supported upgrade. If you have to upgrade from an older version, upgrade to intermediate versions first, or if the system is very out-of-date, consider a complete reinstall. More information on upgrading between releases can be found in the OpenBSD Upgrade Guide 5.0.

- **Shell**: Sometimes, you need to perform repairs or maintenance to a system which will not (or should not) boot to a normal kernel. This option

will allow you to do maintenance to the system. A number of important utilities are available on the boot media.

We are assuming you are choosing "(I)nstall" here.

4.5.2 The Install Questions

Now we start getting the questions that will define how the system is set up. You will note that in most cases, all the questions are asked up front, then the installation takes place. If you have a slow computer or a slow Internet connection, you will be able to answer these questions, walk away, come back later and only have to reboot the system to complete the install.

```
At any prompt except password prompts you can escape to a shell by
typing '!'. Default answers are shown in []'s and are selected by
pressing RETURN.  You can exit this program at any time by pressing
Control-C, but this can leave your system in an inconsistent state.

Choose your keyboard layout ('?' or 'L' for list) [default] Enter
```

In most cases, the default keyboard layout (or terminal type if a serial console install is being done) is appropriate; however don't just take the default, respond appropriately.

```
System hostname? (short form, e.g. 'foo') puffy
```

This value, along with the DNS domain name (specified below), will be saved in the file `/etc/myname`, which is used during normal boot to set the hostname of the system. If you do not set the domain name of the system, the default value of '`my.domain`' will be used.

```
Available network interfaces are: fxp0 vlan0.
Which one do you wish to configure? (or 'done') [fxp0] Enter
```

`vlan0` is the VLAN virtual interface. For our purposes here, we are going to ignore this option and stick to the physical interfaces. If you have multiple physical interfaces, they will be listed here. Note that they are identified by driver name, not generic Ethernet devices. In this case, "`fxp0`" refers to the first device using the `fxp(4)` driver, `fxp1` would be the second device, etc. More on device naming is in FAQ 6.

After selecting the device you wish to configure, you will now configure it. In many cases, you will want to configure it using DHCP:

```
IPv4 address for fxp0? (or 'dhcp' or 'none') [dhcp] Enter
Issuing hostname-associated DHCP request for fxp0.
DHCPDISCOVER on fxp0 to 255.255.255.255 port 67 interval 1
DHCPOFFER from 192.168.1.250 (08:00:20:94:0b:c8)
DHCPREQUEST on fxp0 to 255.255.255.255 port 67
DHCPACK from 192.168.1.250 (08:00:20:94:0b:c8)
bound to 192.168.1.199 -- renewal in 43200 seconds.
```

DHCP will configure the IP address, subnet mask, default gateway, DNS domain name and DNS servers. If you are not using DHCP, you will need to specify all these things manually; see the more detailed discussion below.

If you have any IPv6 configuration to do or there are other interfaces to configure (or you don't like how you configured the previous one), you can do that now, but in our case, we are done:

```
IPv6 address for fxp0? (or 'rtsol' or 'none') [none] Enter
Available network interfaces are: fxp0 vlan0.
Which one do you wish to configure? (or 'done') [done] Enter
Using DNS domainname example.org
Using DNS nameservers at 192.168.1.252
Do you want to do any manual network configuration? [no] Enter
```

If you answer "yes" to the "manual network configuration" question, you will be placed at a shell prompt, where you can configure anything else that needs configuration, then type "exit" to return back to the install program.

```
Password for root account? (will not echo) PaSsWoRd
Password for root account? (again) PaSsWoRd
```

Use a secure password for the root account, remember: on the Internet, they ARE out to get into your computer, they will be trying lots of common passwords people think are really clever.

You will later be given a chance to create an administrative account and disable remote (SSH) access to the root account, but you still want a good password on your root account.

```
Start sshd(8) by default? [yes] Enter
```

Usually, you will want `sshd(8)` running. If your application has no need for `sshd(8)`, there is a small theoretical security advantage to not having it running.

```
Start ntpd(8) by default? [no] y
NTP server? (hostname or 'default') [default] Enter
```

You are here given an option of running OpenNTPD, OpenBSD's NTP implementation. OpenNTPD is a low-impact way of keeping your computer's clock accurately synchronized. The default configuration, using pool.ntp.org, uses a large number of free-access time servers around the world.

One reason you may NOT want to run `ntpd(8)` is if you are running a dual-boot system mostly using another OS which doesn't use a GMT-set hardware clock, as you wouldn't want OpenBSD altering the time for your other OS.

```
Do you expect to run the X Window System? [yes] Enter
```

Not all platforms will ask if you expect to run X, those that do require a sysctl to be set to use X. Answering "y" here will modify `/etc/sysctl.conf` to include the line `machdep.allowaperture=1` or `machdep.allowaperture=2`, depending on your platform.

If you do not intend to run X on this system or are not sure, answer 'N' here, as you can easily change it by editing `/etc/sysctl.conf` and rebooting, should you need to later. There is a potential security advantage to leaving this aperture driver `xf86(4)` disabled, as the graphics engine on a modern video card could potentially be used to alter memory beyond the processor's control. Note that non-graphical applications that require X libraries and utilities to run do NOT need this sysctl to be set.

```
Do you want the X Window System to be started by xdm(1)? [no] y
```

`xdm(1)` starts the X environment at system boot. We'd recommend doing this at install only if you are very confident that X will work on your system by default. Otherwise, configure X before setting up `xdm(1)`.

```
Change the default console to com0? [no] Enter
```

If you wish to configure a serial console rather than your system's default (usually a keyboard and monitor), this is your chance. If you change the default to "y", you will be prompted to set the bit-rate. Note that for serial consoles, faster is not always better, taking your platform's default is highly recommended.

```
Setup a user? (enter a lower-case loginname, or 'no') [no] Enter
```

You are being given an opportunity to create a user OTHER than root for system maintenance. This user will be a member of the "`wheel`" group so they can run `su(1)` and receive mail addressed to root. You will be prompted for a password.

Note that if you wish to create the user, enter the user's name, not "y" or "yes".

```
What timezone are you in? ('?' for list) [Canada/Mountain] US/Michigan
```

OpenBSD assumes your computer's real-time clock (RTC) is set to GMT, but you also have to specify what time zone you are in. There may be several valid answers for your physical location. Hitting "?" at the prompt will help guide you to finding a valid time zone name.

Note that the installer will quite often guess correctly for your time zone, and you can then just hit "Enter".

More on setting the time zone here.

4.5.3 Setting up disks

***Important Note**: Users with a large hard disk (larger than was commonly available when your computer was made) will want to see this section before going any further.*

Laying out your disk appropriately is probably the most difficult part of an OpenBSD install.

Setting up disks in OpenBSD varies a bit between platforms. For i386, amd64, macppc, zaurus and armish, disk setup is done in two stages. First, the OpenBSD slice of the hard disk is defined using `fdisk(8)`, then that slice is subdivided into OpenBSD partitions using `disklabel(8)`.

Some users may be a little confused by the terminology used here. It will appear we are using the word "partition" in two different ways. This observation is correct. There are two layers of partitioning in the above OpenBSD platforms, the first, one could consider the Operating System partitioning, which is how multiple OSs on one computer mark out their own space on the disk, and the second one is how the OpenBSD partition is sub-partitioned into individual filesystems. The first layer is visible as a disk partition to DOS, Windows, and any other OS that uses this disk layout system, the second layer of partitioning is visible only to OpenBSD and those OSs which can directly read an OpenBSD filesystem.

OpenBSD's new installer attempts to make your disk layout tasks easier by having a sane default for "general" use. Note that many people will still want to customize the default, or use their own disk layout, but new users should probably start with this configuration until they see what they need to do differently. Note that the default layout will vary depending on how large your disk system is.

For now, we'll take the defaults on our 40G disk.

```
Available disks are: wd0.
Which one is the root disk? (or 'done') [wd0] Enter
Use DUIDs rather than device names in fstab? [yes] Enter
Disk: wd0       geometry: 5221/255/63 [40960 Megabytes]
Offset: 0       Signature: 0xAA55
            Starting         Ending         LBA Info:
 #: id      C   H   S -      C   H   S [       start:        size ]
--------------------------------------------------------------------------------
 0: 06      0   1   1 -    521 254  63 [          63:     8385867 ] DOS > 32MB
 1: 00      0   0   0 -      0   0   0 [           0:           0 ] unused
 2: 00      0   0   0 -      0   0   0 [           0:           0 ] unused
 3: 00      0   0   0 -      0   0   0 [           0:           0 ] unused
Use (W)hole disk or (E)dit the MBR? [whole] Enter
Setting OpenBSD MBR partition to whole wd0...done.
```

Note that this disk has a pre-existing partition on it – *using "whole" disk will remove it!*.

Setting up the "whole" disk for OpenBSD does a number of important things:

- erases any existing partitions on the disk
- creates an MBR and disk signature so the disk can be booted
- Creates an OpenBSD partition using the entire disk
- Sets that partition as "active".

There are many times when you won't want to do that, including:

- You wish to retain other OS partitions
- You wish to retain "setup", "suspend to disk", or other system partitions
- You wish to build a multi-booting system

Note that it is critical that a new (or never-used for booting) drive has a valid MBR, a valid signature, an OpenBSD partition, and a partition flagged as "active". If you don't do these things using the "Use whole disk" option, you need to make sure they get done manually. More information on fdisk partitioning your disk below.

Now we will break up our OpenBSD fdisk partition into OpenBSD disk partitions using `disklabel(8)`:

```
Setting OpenBSD MBR partition to whole wd0...done.
The auto-allocated layout for wd0 is:
#                size           offset  fstype [fsize bsize  cpg]
  a:          1024.0M               64  4.2BSD   2048 16384    1 # /
  b:           199.0M          2097216    swap
  c:         40960.0M                0  unused
  d:          2822.9M          2504768  4.2BSD   2048 16384    1 # /tmp
  e:          4295.0M          8286112  4.2BSD   2048 16384    1 # /var
  f:          2048.0M         17082240  4.2BSD   2048 16384    1 # /usr
  g:          1024.0M         21276544  4.2BSD   2048 16384    1 # /usr/X11R6
  h:          5426.7M         23373696  4.2BSD   2048 16384    1 # /usr/local
  i:          1699.7M         34487520  4.2BSD   2048 16384    1 # /usr/src
  j:          2048.0M         37968576  4.2BSD   2048 16384    1 # /usr/obj
  k:         20367.4M         42162880  4.2BSD   2048 16384    1 # /home
Use (A)uto layout, (E)dit auto layout, or create (C)ustom layout? [a] Enter
```

The installer has presented us with its proposed "Auto layout" for OpenBSD partitions on our disk, which we are going to accept.

If the proposed layout is not appropriate for your needs, you can, of course, edit the default or customize it completely, more details on the disklabel partitioning below.

NOTE for re-installers: The new installer will not clear your old disklabel if you chose "(C)ustom Layout", but you will need to re-specify each mount point using the 'm' option in disklabel(8).

The installer now creates those partitions and creates file systems on them using **newfs(8)**, and mounts them for installation:

```
/dev/rwd0a: 1024.0MB in 2097152 sectors of 512 bytes
6 cylinder groups of 202.47MB, 12958 blocks, 25984 inodes each
/dev/rwd0k: 20367.4MB in 41712448 sectors of 512 bytes
101 cylinder groups of 202.47MB, 12958 blocks, 25984 inodes each
/dev/rwd0d: 2822.9MB in 5781344 sectors of 512 bytes
14 cylinder groups of 202.47MB, 12958 blocks, 25984 inodes each
/dev/rwd0f: 2048.0MB in 4194304 sectors of 512 bytes
11 cylinder groups of 202.47MB, 12958 blocks, 25984 inodes each
/dev/rwd0g: 1024.0MB in 2097152 sectors of 512 bytes
6 cylinder groups of 202.47MB, 12958 blocks, 25984 inodes each
/dev/rwd0h: 5426.7MB in 11113824 sectors of 512 bytes
27 cylinder groups of 202.47MB, 12958 blocks, 25984 inodes each
/dev/rwd0j: 2048.0MB in 4194304 sectors of 512 bytes
11 cylinder groups of 202.47MB, 12958 blocks, 25984 inodes each
/dev/rwd0i: 1699.7MB in 3481056 sectors of 512 bytes
9 cylinder groups of 202.47MB, 12958 blocks, 25984 inodes each
/dev/rwd0e: 4295.0MB in 8796128 sectors of 512 bytes
22 cylinder groups of 202.47MB, 12958 blocks, 25984 inodes each
/dev/wd0a on /mnt type ffs (rw, asynchronous, local)
/dev/wd0k on /mnt/home type ffs (rw, asynchronous, local, nodev, nosuid)
/dev/wd0d on /mnt/tmp type ffs (rw, asynchronous, local, nodev, nosuid)
/dev/wd0f on /mnt/usr type ffs (rw, asynchronous, local, nodev)
/dev/wd0g on /mnt/usr/X11R6 type ffs (rw, asynchronous, local, nodev)
/dev/wd0h on /mnt/usr/local type ffs (rw, asynchronous, local, nodev)
/dev/wd0j on /mnt/usr/obj type ffs (rw, asynchronous, local, nodev, nosuid)
/dev/wd0i on /mnt/usr/src type ffs (rw, asynchronous, local, nodev, nosuid)
/dev/wd0e on /mnt/var type ffs (rw, asynchronous, local, nodev, nosuid)
```

You will note there is a *c* partition we seem to have ignored. This partition is your entire hard disk; don't attempt to alter it.

4.5.4 Choosing installation media and file sets

Next, you will get a chance to choose your installation media. In this case, we will install from an FTP server.

```
Location of sets? (cd disk ftp http or 'done') [cd] ftp
HTTP/FTP proxy URL? (e.g. 'http://proxy:8080', or 'none') [none] Enter
Server? (hostname, list#, 'done' or '?') [mirror.example.org] obsd.cec.mtu.edu
```

If you can't remember your favorite (or any!) mirror's location, the installer will often be able to come up with a default of a mirror which will work well for you. Otherwise, hit "?" to have a list of mirrors displayed, and select the number of a mirror that will work well for you.

```
Server directory? [pub/OpenBSD/5.0/i386] Enter
Login? [anonymous] Enter
```

The public FTP mirrors all support anonymous downloads, of course, but you may have a local machine which requires a login and password.

You can now adjust the list of file sets.

```
Select sets by entering a set name, a file name pattern or 'all'. De-select
sets by prepending a '-' to the set name, file name pattern or 'all'. Selected
sets are labelled '[X]'.
    [X] bsd           [X] etc50.tgz     [X] xbase50.tgz   [X] xserv50.tgz
    [X] bsd.rd        [X] comp50.tgz    [X] xetc50.tgz
    [ ] bsd.mp        [X] man50.tgz     [X] xshare50.tgz
    [X] base50.tgz    [X] game50.tgz    [X] xfont50.tgz
Set name(s)? (or 'abort' or 'done') [done] Enter
```

At a bare minimum, you need to have a kernel (`bsd`), the `base50.tgz` and `etc50.tgz` file sets. Unless you know what you are doing, stick with the default sets. You can add and remove file sets using "+" and "-" chars in front of the file set name, and also use wildcards:

- **`-comp50.tgz`** removes `comp50.tgz`
- **`+bsd.mp`** adds `bsd.mp`
- **`-x*`** removes all X components

But again, we'll take the default. This machine is a single-processor system, so `bsd.mp` is not installed, but everything else is. If it could later be upgraded to a multi-processor system, you might want to install `bsd.mp` as well.

And now, we start our install! This is the point at which you might want to come back later if you have a slow computer or Internet connection, though with a fast computer and local files, this process may take just a couple minutes or less!

```
bsd          100% |*************************************|  8764 KB    00:05
bsd.rd       100% |*************************************|  6268 KB    00:03
base50.tgz   100% |*************************************| 53928 KB    00:26
etc50.tgz    100% |*************************************|   513 KB    00:00
comp50.tgz   100% |*************************************| 57224 KB    00:28
man50.tgz    100% |*************************************|  9482 KB    00:06
game50.tgz   100% |*************************************|  2568 KB    00:02
xbase50.tgz  100% |*************************************| 11331 KB    00:06
xetc50.tgz   100% |*************************************| 71741       00:00
xshare50.tgz 100% |*************************************|  3357 KB    00:04
xfont50.tgz  100% |*************************************| 38868 KB    00:17
xserv50.tgz  100% |*************************************| 31205 KB    00:15
Location of sets? (cd disk ftp http or 'done') [done] Enter
```

Yes, it is asking us again where we wish to install things from. This is so either missed, forgotten or failed file sets can be re-installed, and also so custom file sets can be installed.

Again, we just take the default, we are done installing files

```
Saving configuration files...done.
Generating initial host.random file...done.
Install non-free firmware files on first boot? [no] Enter
```

Some systems have devices which require firmware which can not be distributed as part of OpenBSD for licensing reasons. If you chose to install the non-free firmware on first boot, the first time the system is booted, the system will attempt to connect to the Internet and download the available restricted firmware files.

```
Making all device nodes...done.
CONGRATULATIONS! Your OpenBSD install has been successfully completed!
To boot the new system, enter 'reboot' at the command prompt.
When you login to your new system the first time, please read your mail
using the 'mail' command.

#
```

4.5.5 First boot!

OpenBSD is now installed on your system and ready for its first boot, but before you do...

Before you reboot

At this point, your system is installed and ready to be rebooted and configured for service. Before doing this, however, it would be wise to check out the Errata page to see if there are any bugs that would immediately impact you.

After you reboot

On your first boot, SSH keys will be generated. On modern computers, this will take a few seconds, you may not even notice it happening. On older systems, it may take many minutes, potentially even an hour or more for really slow systems. One of your first things to read after you install your system is afterboot(8).

You may also find the following links useful:

- Adding users in OpenBSD
- Initial Network Setup
- Man Pages of popular/useful commands
- OpenBSD man pages on the Web
- The OpenBSD Packages and Ports system for installing software

4.5.6 One last thing...

The OpenBSD developers ask you to Send in a copy of your dmesg. This is really appreciated by the developers, and ultimately, all users.

4.6 Details for a more complex install

Sometimes you can't just take the defaults. Here are some more details on parts of the installation process.

4.6.1 Setting up the network

If you don't have a DHCP server available, you will have to set up your network adapter(s) manually. Here's an example:

```
Which one do you wish to configure? (or 'done') [xl0] Enter
IPv4 address for xl0? (or 'dhcp' or 'none') [dhcp] 192.168.1.37
Netmask? [255.255.255.0] 255.255.254.0
IPv6 address for xl0? (or 'rtsol' or 'none') [none] Enter
```

After that set of questions, you will be given a chance to configure any other network adapters that your machine has. If you specify another network adapter here, the above questions repeat.

```
Available network interfaces are: xl0 vlan0.
Which one do you wish to configure? (or 'done') [done]
```

Now, you will set up the default gateway and DNS servers, things that impact all network adapters:

```
Default IPv4 route? (IPv4 address, 'dhcp' or 'none') 192.168.1.1
add net default: gateway 192.168.1.1
DNS domain name? (e.g. 'bar.com') [my.domain] example.org
DNS nameservers? (IP address list or 'none') [none] 192.168.1.250 192.168.1.251
```

Note that multiple DNS servers can be listed, separated by spaces.

Sometimes, you will have to do something more, for example set up a wireless access key or hard-set a duplex or speed setting (don't do this unless you absolutely HAVE to, fixing your switch configuration is a much better idea!). You are now given a chance to drop to the shell and do any manual configuration that you would like.

```
Do you want to do any manual network configuration? [no] y
Type 'exit' to return to install.
# ifconfig xl0 media
xl0: flags=8843<UP,BROADCAST,RUNNING,SIMPLEX,MULTICAST> mtu 1500
        lladdr 00:08:74:2c:df:9c
        groups: egress
        media: Ethernet autoselect (100baseTX full-duplex)
        status: active
        supported media:
                media 10baseT
```

```
                media 10baseT mediaopt full-duplex
                media 100baseTX
                media 100baseTX mediaopt full-duplex
                media autoselect
        inet 192.168.1.37 netmask 0xfffffe00 broadcast 192.168.1.255
# ifconfig xl0 media 100baseTX mediaopt full-duplex
# ifconfig xl0
xl0: flags=8843<UP,BROADCAST,RUNNING,SIMPLEX,MULTICAST> mtu 1500
        lladdr 00:08:74:2c:df:9c
        groups: egress
        media: Ethernet 100baseTX full-duplex
        status: active
        inet6 fe80::208:74ff:fe2c:df9c%xl0 prefixlen 64 scopeid 0x1
        inet 192.168.1.37 netmask 0xfffffe00 broadcast 192.168.1.255
# exit
...setup resumes...
```

(back to where we might have been)

4.6.2 Setting the Time Zone

Time in Unix is not a simple thing (or put another way, time in Unix is a *really* simple thing, human time is a politically manipulated mess). Time zone files help the system convert Unix time (the number of seconds past midnight GMT, Jan 1, 1970) to human time, taking into account things like time zones, daylight savings time (DST), DST rule changes, etc. They also include the history of changes. Multiple time zone definition files will sometimes give the same *current* time, but may have different history. For example, EST5EDT and US/Michigan have the same time NOW, but back in 1975, the rules were different, so if you were doing math with dates and times that involved 1975, you would care about the differences. You should use the most specific and accurate timezone file you can for your region, rather than one that just gives the correct time at this moment.

OpenBSD's installer will help you find an appropriate time zone file for you if you are not sure. Simply hit "?" at each prompt, and the installer will show you options. If the first level of answers doesn't suite you, pick a continent or country, and look at your options there:

```
What timezone are you in? ('?' for list) [right/EST5EDT] ?
Africa/      Chile/       GB-Eire      Israel       NZ-CHAT      UCT
America/     Cuba         GMT          Jamaica      Navajo       US/
Antarctica/  EET          GMT+0        Japan        PRC          UTC
Arctic/      EST          GMT-0        Kwajalein    PST8PDT      Universal
Asia/        EST5EDT      GMT0         Libya        Pacific/     W-SU
Atlantic/    Egypt        Greenwich    MET          Poland       WET
Australia/   Eire         HST          MST          Portugal     Zulu
```

```
Brazil/      Etc/         Hongkong     MST7MDT      ROC          posix/
CET          Europe/      Iceland      Mexico/      ROK          posixrules
CST6CDT      Factory      Indian/      Mideast/     Singapore    right/
Canada/      GB           Iran         NZ           Turkey
What timezone are you in? ('?' for list) [right/EST5EDT] US
What sub-timezone of 'US' are you in? ('?' for list) ?
Alaska          Central         Hawaii          Mountain        Samoa
Aleutian        East-Indiana    Indiana-Starke  Pacific
Arizona         Eastern         Michigan        Pacific-New
What sub-timezone of 'US' are you in? ('?' for list) Michigan
```

We've now set the time to "US/Michigan". This creates a symbolic link in `/etc` pointing to the appropriate zoneinfo file in `/usr/share/zoneinfo`, something like this:

`/etc/localtime -> /usr/share/zoneinfo/US/Michigan`

Note the directory "`right/`", this directory includes leap second adjustments, but otherwise duplicates the standard zoneinfo choices. More here.

(back to where we might have been)

4.6.3 Custom fdisk(8) layout

Note: only some OpenBSD platforms use fdisk at all, and usually, only i386 and amd64 users will have to worry about getting fancy with `fdisk`. Users of most other `fdisk(8)` using platforms generally don't have to worry about multibooting or setup/diagnostic partitions. For this reason, this section is focused on i386 and amd64. `fdisk(8)` is used to mark off the OpenBSD part of your hard disk. It helps mark off the part of the disk used by OpenBSD from the parts used by other OSs or system functions.

If you have a partition on your disk you wish to retain or wish to leave space for another partition, you will NOT want to chose "(W)hole disk", but will need to edit the partition table with `fdisk(8)`. More information on manually running `fdisk(8)` can be found here. Before working with any system that has data you don't wish to lose, make sure you have a good backup. It is very easy in this process to clobber important data, so make sure you are ready to get it back, if need be.

If you are adding OpenBSD to an existing system, you will probably need to create some free space on your system before installing OpenBSD. This will usually involve deleting or possibly reducing the size of existing partitions. The program gparted has been found useful for shrinking the partitions of many popular OSs, making it possible to install OpenBSD on the freed space.

In this example, we will assume we are starting with a blank 40G disk and wish to create a multi-boot system, reserving 5G at the beginning of the disk for Windows, and the rest for OpenBSD. Note that a blank drive has to have valid MBR boot code and signature written to the disk before it can be booted.

The process is very much the same for working around an existing partition, you just need to skip over the parts where we create the Windows partition and

worry about installing the MBR boot code.

```
Available disks are: wd0.
Which one is the root disk? (or 'done') [wd0] Enter
MBR has invalid signature; not showing it.
```

IF the disk had a valid MBR in place, it would show you the existing partition table, which can be a good way to show if a disk may have data on it already.

```
Use (W)hole disk or (E)dit the MBR? [whole] e

You will now create a single MBR partition to contain your OpenBSD data. This
partition must have an id of 'A6'; must *NOT* overlap other partitions; and
must be marked as the only active partition.  Inside the fdisk command, the
'manual' command describes all the fdisk commands in detail.

Disk: wd0       geometry: 4998/255/63 [80293248 Sectors]
Offset: 0       Signature: 0x0
            Starting         Ending         LBA Info:
 #: id      C   H   S -      C   H   S [       start:        size ]
--------------------------------------------------------------------------------
 0: 00      0   0   0 -      0   0   0 [           0:           0 ] unused
 1: 00      0   0   0 -      0   0   0 [           0:           0 ] unused
 2: 00      0   0   0 -      0   0   0 [           0:           0 ] unused
 3: 00      0   0   0 -      0   0   0 [           0:           0 ] unused
Enter 'help' for information
fdisk: 1>
```

First of all notice the fdisk prompt. The number "1" indicates the first level of partition tables – if you were editing an extended partition, it would be "2" (or bigger). Extended partitions are partitions which have their own sub-partition table, getting around the IBM AT four partition design limit. Extended partitions won't be covered here.

First, we will make partition "0" a 5G Windows partition (using NTFS), and partition "1" will be our OpenBSD partition using the rest of the disk.

```
fdisk: 1> e 0
            Starting         Ending         LBA Info:
 #: id      C   H   S -      C   H   S [       start:        size ]
--------------------------------------------------------------------------------
 0: 00      0   0   0 -      0   0   0 [           0:           0 ] unused
Partition id ('0' to disable)  [0 - FF]: [0] (? for help)
```

Since we don't know by memory what the partition ID is for NTFS, we hit "?" here to get a list.

```
Partition id ('0' to disable)  [0 - FF]: [0] (? for help) ?
```

```
Choose from the following Partition id values:
00 unused         20 Willowsoft     66 NetWare 386    A9 NetBSD
01 DOS FAT-12     24 NEC DOS        67 Novell         AB MacOS X boot
02 XENIX /        27 Win Recovery   68 Novell         AF MacOS X HFS+
03 XENIX /usr     38 Theos          69 Novell         B7 BSDI filesy*
04 DOS FAT-16     39 Plan 9         70 DiskSecure     B8 BSDI swap
05 Extended DOS   40 VENIX 286      75 PCIX           BF Solaris
06 DOS > 32MB     41 Lin/Minux DR   80 Minix (old)    C0 CTOS
07 NTFS           42 LinuxSwap DR   81 Minix (new)    C1 DRDOSs FAT12
08 AIX fs         43 Linux DR       82 Linux swap     C4 DRDOSs < 32M
09 AIX/Coherent   4D QNX 4.2 Pri    83 Linux files*   C6 DRDOSs >=32M
0A OS/2 Bootmgr   4E QNX 4.2 Sec    84 OS/2 hidden    C7 HPFS Disbled
0B Win95 FAT-32   4F QNX 4.2 Ter    85 Linux ext.     DB CPM/C.DOS/C*
0C Win95 FAT32L   50 DM             86 NT FAT VS      DE Dell Maint
0E DOS FAT-16     51 DM             87 NTFS VS        E1 SpeedStor
0F Extended LBA   52 CP/M or SysV   8E Linux LVM      E3 SpeedStor
10 OPUS           53 DM             93 Amoeba FS      E4 SpeedStor
11 OS/2 hidden    54 Ontrack        94 Amoeba BBT     EB BeOS/i386
12 Compaq Diag.   55 EZ-Drive       99 Mylex          EE EFI GPT
14 OS/2 hidden    56 Golden Bow     9F BSDI           EF EFI Sys
16 OS/2 hidden    5C Priam          A0 NotebookSave   F1 SpeedStor
17 OS/2 hidden    61 SpeedStor      A5 FreeBSD        F2 DOS 3.3+ Sec
18 AST swap       63 ISC, HURD, *   A6 OpenBSD        F4 SpeedStor
19 Willowtech     64 NetWare 2.xx   A7 NEXTSTEP       FF Xenix BBT
1C ThinkPad Rec   65 NetWare 3.xx   A8 MacOS X
Partition id ('0' to disable)  [0 - FF]: [0] (? for help) 07
```

Now we define its starting and ending points:

```
Do you wish to edit in CHS mode? [n]
```

CHS mode allows you to specify disk space in Cylinders, Heads and Sectors. Keep in mind that for modern hard disks, the CHS numbers are completely bogus, just three numbers that translate to a sector on the disk, which is translated to your drives physical geometry (which probably varies across the disk anyway).

If you answer "y" here, you will be prompted for the starting and ending cylinder, head and sector. If you answer "no" here (as we will), you will be prompted for starting sector and the size. Editing by CHS is often easier when working around an existing partition, starting sector and size is often easier when you want to quickly create a partition of a given size.

```
offset: [0] 64
```

The fdisk platforms need gap before the first partition. The exact amount will not matter on modern machines, OpenBSD defaults to 64 sectors. This is recommended for performance reasons on modern disks, and does not matter on older disks.

```
size: [0] 5g
Rounding to nearest cylinder: 10490382
```

The "Size" value can be the number of sectors (512 bytes each), or the desired capacity when followed by a "k", "m" or "g". When editing using offset and size, fdisk will round your partition so it ends on a cylinder boundary (OpenBSD doesn't care about this, and it is possible no modern OS cares about this, but some might have at one time).

Now, let's look at our new partition:

```
fdisk:*1> p
Disk: wd0       geometry: 4998/255/63 [80293248 Sectors]
Offset: 0       Signature: 0x0
            Starting         Ending         LBA Info:
 #: id      C   H   S -      C   H   S [       start:        size ]
-------------------------------------------------------------------------------
 0: 07      0   1   1 -    652 254  63 [          63:    10490382 ] NTFS
 1: 00      0   0   0 -      0   0   0 [           0:           0 ] unused
 2: 00      0   0   0 -      0   0   0 [           0:           0 ] unused
 3: 00      0   0   0 -      0   0   0 [           0:           0 ] unused
fdisk:*1>
```

Note that the prompt now includes an "*", this means there are unsaved changes.

We've now created our Windows partition. Note that this partition is so far just reserved space on the disk, it isn't formatted; no file system exists here. You will worry about that when you install Windows; we've accomplished our goal of reserving space for the Windows partition to be created later.

Now we create our OpenBSD partition. In this case, the partition ID will be "A6".

```
fdisk:*1> e 1
            Starting         Ending         LBA Info:
 #: id      C   H   S -      C   H   S [       start:        size ]
-------------------------------------------------------------------------------
 1: 00      0   0   0 -      0   0   0 [           0:           0 ] unused
Partition id ('0' to disable)  [0 - FF]: [0] (? for help) a6
Do you wish to edit in CHS mode? [n] Enter
offset: [0]
```

uh-oh! What's our offset? Simple – the offset of the previous partition plus the size of the partition, in this case, 63+10490382 = 10490445.

```
offset: [0] 10490445
size: [0] *
fdisk:*1>
```

Note that here, we entered "*" as the size, meaning "rest of the disk". Again, we could have entered the size in sectors, "m" or "g" if we wanted to leave space for something else.

Now we look at our partition table:

```
fdisk:*1> p
Disk: wd0       geometry: 4998/255/63 [80293248 Sectors]
Offset: 0       Signature: 0x0
            Starting         Ending         LBA Info:
 #: id      C   H   S -      C   H   S [       start:        size ]
--------------------------------------------------------------------------------
 0: 07      0   1   1 -    652 254  63 [          63:    10490382 ] NTFS
 1: A6    653   0   1 -   4998   5  63 [    10490445:    69802803 ] OpenBSD
 2: 00      0   0   0 -      0   0   0 [           0:           0 ] unused
 3: 00      0   0   0 -      0   0   0 [           0:           0 ] unused
fdisk:*1>
```

WE AREN'T DONE YET! This disk is not yet bootable! As it was a brand new disk, the disk's MBR was completely blank. The "`Signature: 0x0`" message there shows there is not a valid signature (`0xAA55`), which indicates there definitely is not a valid boot code. Of course, you could have a valid signature without valid boot code, through either random bad luck or damage to the existing boot code, but an invalid signature pretty well indicates you are lacking boot code, so we will install it now using the "`update`" command:

```
fdisk:*1> update
Machine code updated.
fdisk:*1>
```

We also have to "`flag`" a partition as "`active`" so the boot ROM knows what partition to boot from:

```
fdisk:*1> f 1
Partition 1 marked active.
```

Now, let's see how it looks:

```
fdisk:*1> p
Disk: wd0       geometry: 4998/255/63 [80293248 Sectors]
Offset: 0       Signature: 0xAA55
            Starting         Ending         LBA Info:
 #: id      C   H   S -      C   H   S [       start:        size ]
--------------------------------------------------------------------------------
 0: 07      0   1   1 -    652 254  63 [          63:    10490382 ] NTFS
*1: A6    653   0   1 -   4998   5  63 [    10490445:    69802803 ] OpenBSD
 2: 00      0   0   0 -      0   0   0 [           0:           0 ] unused
 3: 00      0   0   0 -      0   0   0 [           0:           0 ] unused
fdisk:*1>
```

A checklist of things you want to make sure about before you exit `fdisk(8)`:

- Valid signature?
- non-overlapping partitions?
- OpenBSD partition with an "A6" id?
- Proper partition (probably OpenBSD) flagged active?

(Back to where we may have been)

4.6.4 Custom disklabel layout

Inside the OpenBSD `fdisk(8)` partition, we use disklabel(8) to create OpenBSD file system partitions. OpenBSD labels its file system partitions using sixteen letters, "a" through "p". Partition "a" on the boot disk is defined as the root partition, "b" on the boot disk is the default swap partition. "c" on all disks is the "whole disk" partition, it is used by programs that have to have raw access to the physical disk, such as `fdisk(8)` and `disklabel(8)`. The "c" partition is created automatically for you, and should not be deleted or changed. The remaining letters are available for you to define mount points on. You may skip letters, you can define them in any order, and they can be in any order on the disk (although some platforms do have a requirement for where the "a" partition is). You can also leave gaps in the disk that are unallocated, and allocate them later, or potentially enlarge existing partitions later into that unallocated space using `growfs(8)`. All partitions which have native FFS partitions on them should be within the OpenBSD `fdisk(8)` partition, however non-OpenBSD partitions can (and usually should) be outside the OpenBSD fdisk partition.

More information on using disklabel can be found here.

More information on the why partitioning is good and strategy for a good partitioning plan are below.

The OpenBSD installer will attempt to auto-partition your disk in a usable, "general purpose" configuration, based on the size of your disk. If your disk is big enough, unused space will be allocated to the `/home` partition. While this is often quite useful, it doesn't satisfy all users' needs.

For our example, we'll assume we are building a static web server for some of our friends to use. We have a machine attached to a modest Internet connection, with a 40G disk, with most of it used for OpenBSD (with the same 5G Windows partition as the example above. Why? Maybe this system has a RAID controller which is supported by OpenBSD, but manageable only from within Windows. More likely, because the FAQ editor doesn't feel like maintaining lots of different example systems).

The web pages served by an OpenBSD web server will be in `/var/www`, and very little will be stored in `/home`, so this indicates a definite change from the default that needs to be made. For the sake of discussion, we'll also assume that we won't need to rebuild the OS from source on this machine (we'll do

that elsewhere). The system will not run X, however being that some web applications expect X to be installed, we will have X installed. The machine is not overly powerful, it can't have more than 1G RAM in it, and it is unlikely our application will ever desire more than that.

So, after a bit of thought, our plan is to partition the system like this:

- `/ - root`: 100m. This will be 'a'.
- `swap`: 1G (so we'll always have enough space for a core dump), this will be partition 'b'
- `/usr`: 2g, partition d
- `/tmp`: 100m (we don't anticipate much use of this), partition e
- `/usr/local`: 2g, partition f
- `/usr/X11R6`: 1g, partition g
- `/home`: 1g, partition h
- `/var`: 1g (that's a lot of system log files), partition j
- `/var/www`: rest of disk, partition k

```
The auto-allocated layout for wd0 is:
#                size           offset  fstype [fsize bsize  cpg]
  a:           1024.0M         10490445  4.2BSD   2048 16384    1 # /
  b:            252.1M         12587597    swap
  c:          39205.7M                0  unused
  d:           2319.3M         13103933  4.2BSD   2048 16384    1 # /tmp
  e:           3653.9M         17853877  4.2BSD   2048 16384    1 # /var
  f:           1149.8M         25337016  4.2BSD   2048 16384    1 # /usr
  g:           1024.0M         27691862  4.2BSD   2048 16384    1 # /usr/X11R6
  h:           3422.6M         29789014  4.2BSD   2048 16384    1 # /usr/local
  i:           5122.3M               63    NTFS
  j:           1848.7M         36798433  4.2BSD   2048 16384    1 # /usr/src
  k:           1848.7M         40584654  4.2BSD   2048 16384    1 # /usr/obj
  l:          17540.2M         44370875  4.2BSD   2048 16384    1 # /home
Use (A)uto layout, (E)dit auto layout, or create (C)ustom layout? [a] c
```

If we had only minor revisions, we'd probably opt to "Edit" the custom layout rather than starting from a clean slate, but we are going to do things the hard way here.

```
You will now create an OpenBSD disklabel inside the OpenBSD MBR
partition. The disklabel defines how OpenBSD splits up the MBR partition
into OpenBSD partitions in which filesystems and swap space are created.
You must provide each filesystem's mountpoint in this program.
```

```
The offsets used in the disklabel are ABSOLUTE, i.e. relative to the
start of the disk, NOT the start of the OpenBSD MBR partition.

Label editor (enter '?' for help at any prompt)
> p
OpenBSD area: 10490445-80293248; size: 69802803; free: 69802803
#                size           offset  fstype [fsize bsize  cpg]
  c:         80293248                0  unused
  i:         10490382               63    NTFS
>
```

Note there are already two partitions here – the "c" partition which is always there and created for you, but `disklabel(8)` has also noticed the existing NTFS partition and assigned it a disklabel partition so it could potentially be accessed by OpenBSD (note, at this time, NTFS support is experimental and requires a custom kernel but FAT/FAT32 support is quite good).

We will now create our partitions. We will start with the "a" partition, our root partition:

```
> a a
offset: [10490445] Enter
size: [69802803] 100m
Rounding to cylinder: 208845
FS type: [4.2BSD]  Enter
mount point: [none] /
>
```

Note that disklabel defaulted to the first available OpenBSD sector on the disk, which is what we want. It also defaulted to a size of all available space, which is NOT what we want. Here we overrode it with our preferred size, which can be specified in sectors, "M" or "G".

You will usually want to use the default FS type of "`4.2BSD`" for a FFS (Fast File System) or FFS2 partition, though other types you may find useful include "`swap`" and "`RAID`".

Finally is the mount point. Our "a" partition is the root partition, by definition.

Now, we do swap, which is our 'b' partition (again, this is a requirement – 'b' on your boot disk is swap):

```
> a b
offset: [10699290] Enter
size: [69593958] 1g
Rounding to cylinder: 2104515
FS type: [swap] Enter
>
```

Again, disklabel correctly calculated our starting sector, and presented us with a suggested size of "entire remaining space", which we again overrode with our desired size. Since this is the 'b' partition, disklabel assumed it was to be used for swap, and when we confirmed that, it didn't bother to ask us a mount point.

We are now ready to create the rest of the partitions.

```
> a d
offset: [12803805] Enter
size: [67489443] 2g
Rounding to cylinder: 4209030
FS type: [4.2BSD] Enter
mount point: [none] /usr
> a e
offset: [17012835] Enter
size: [63280413] 100m
Rounding to cylinder: 208845
FS type: [4.2BSD] Enter
mount point: [none] /tmp
> a f
offset: [17221680] Enter
size: [63071568] 2g
Rounding to cylinder: 4209030
FS type: [4.2BSD] Enter
mount point: [none] /usr/local
> a g
offset: [21430710] Enter
size: [58862538] 1g
Rounding to cylinder: 2104515
FS type: [4.2BSD] Enter
mount point: [none] /usr/X11R6
> a h
offset: [23535225] Enter
size: [56758023] 1g
Rounding to cylinder: 2104515
FS type: [4.2BSD] Enter
mount point: [none] /home
> a j
offset: [25639740] Enter
size: [54653508] 1g
Rounding to cylinder: 2104515
FS type: [4.2BSD] Enter
mount point: [none] /var
> a k
offset: [27744255] Enter
size: [52548993] Enter
```

```
FS type: [4.2BSD] Enter
mount point: [none] /var/www
>
```

Note that on the /var/www partition ("k"), we just took the default to use all remaining available disk space. With modern monstrously huge drives, this is usually a bad idea. If you know you will never use it, don't allocate it, and save it for some future use.

Now, let's look at our results, using the "p" and "p m" commands:

```
> p
OpenBSD area: 10490445-80293248; size: 69802803; free: 0
#                size           offset  fstype [fsize bsize  cpg]
  a:           208845         10490445  4.2BSD   2048 16384    1 # /
  b:          2104515         10699290    swap
  c:         80293248                0  unused
  d:          4209030         12803805  4.2BSD   2048 16384    1 # /usr
  e:           208845         17012835  4.2BSD   2048 16384    1 # /tmp
  f:          4209030         17221680  4.2BSD   2048 16384    1 # /usr/local
  g:          2104515         21430710  4.2BSD   2048 16384    1 # /usr/X11R6
  h:          2104515         23535225  4.2BSD   2048 16384    1 # /home
  i:         10490382               63    NTFS
  j:          2104515         25639740  4.2BSD   2048 16384    1 # /var
  k:         52548993         27744255  4.2BSD   2048 16384    1 # /var/www
> p m
OpenBSD area: 10490445-80293248; size: 34083.4M; free: 0.0M
  #                size           offset  fstype [fsize bsize  cpg]
  a:            102.0M         10490445  4.2BSD   2048 16384    1 # /
  b:           1027.6M         10699290    swap
  c:          39205.7M                0  unused
  d:           2055.2M         12803805  4.2BSD   2048 16384    1 # /usr
  e:            102.0M         17012835  4.2BSD   2048 16384    1 # /tmp
  f:           2055.2M         17221680  4.2BSD   2048 16384    1 # /usr/local
  g:           1027.6M         21430710  4.2BSD   2048 16384    1 # /usr/X11R6
  h:           1027.6M         23535225  4.2BSD   2048 16384    1 # /home
  i:           5122.3M               63    NTFS
  j:           1027.6M         25639740  4.2BSD   2048 16384    1 # /var
  k:          25658.7M         27744255  4.2BSD   2048 16384    1 # /var/www
>
```

Like with fdisk, you don't want your OpenBSD disklabel partitions to overlap (other than the 'c' partition, which overlaps everything, of course).

Write your changes and quit disklabel:

```
> w
> q
No label changes.
```

```
  newfs: reduced number of fragments per cylinder group from 13048 to 12992 to
  enlarge last cylinder group
  /dev/rwd0a: 102.0MB in 208844 sectors of 512 bytes
  5 cylinder groups of 25.38MB, 1624 blocks, 3328 inodes each
  /dev/rwd0h: 1027.6MB in 2104512 sectors of 512 bytes
  6 cylinder groups of 202.47MB, 12958 blocks, 25984 inodes each
  newfs: reduced number of fragments per cylinder group from 13048 to 12992 to
  enlarge last cylinder group
  /dev/rwd0e: 102.0MB in 208844 sectors of 512 bytes
  5 cylinder groups of 25.38MB, 1624 blocks, 3328 inodes each
  /dev/rwd0d: 2055.2MB in 4209028 sectors of 512 bytes
  11 cylinder groups of 202.47MB, 12958 blocks, 25984 inodes each
  /dev/rwd0g: 1027.6MB in 2104512 sectors of 512 bytes
  6 cylinder groups of 202.47MB, 12958 blocks, 25984 inodes each
  /dev/rwd0f: 2055.2MB in 4209028 sectors of 512 bytes
  11 cylinder groups of 202.47MB, 12958 blocks, 25984 inodes each
  /dev/rwd0j: 1027.6MB in 2104512 sectors of 512 bytes
  6 cylinder groups of 202.47MB, 12958 blocks, 25984 inodes each
  /dev/rwd0k: 25658.7MB in 52548992 sectors of 512 bytes
  127 cylinder groups of 202.47MB, 12958 blocks, 25984 inodes each
  /dev/wd0a on /mnt type ffs (rw, asynchronous, local)
  /dev/wd0h on /mnt/home type ffs (rw, asynchronous, local, nodev, nosuid)
  /dev/wd0e on /mnt/tmp type ffs (rw, asynchronous, local, nodev, nosuid)
  /dev/wd0d on /mnt/usr type ffs (rw, asynchronous, local, nodev)
  /dev/wd0g on /mnt/usr/X11R6 type ffs (rw, asynchronous, local, nodev)
  /dev/wd0f on /mnt/usr/local type ffs (rw, asynchronous, local, nodev)
  /dev/wd0j on /mnt/var type ffs (rw, asynchronous, local, nodev, nosuid)
  /dev/wd0k on /mnt/var/www type ffs (rw, asynchronous, local, nodev, nosuid)

  Let's install the sets!
...
```

(Back to where we may have been)

4.7 What files are needed for installation?

The complete OpenBSD installation is broken up into a number of separate `file sets`. Not every application requires every file set, however new users are recommended to install ALL of them. Here is an overview of each:

- **`bsd`** - This is the Kernel. **Required**
- **`bsd.mp`** - Multi-processor (SMP) kernel (only some platforms)
- **`bsd.rd`** - RAM disk kernel
- **`base50.tgz`** - Contains the base OpenBSD system **Required**

- **etc50.tgz** - Contains all the files in /etc **Required**
- **comp50.tgz** - Contains the compiler and its tools, headers and libraries.
- **man50.tgz** - Contains man pages
- **game50.tgz** - Contains the games for OpenBSD
- **xbase50.tgz** - Contains the base libraries and utilities for X11
- **xetc50.tgz** - Contains the /etc/X11 and /etc/fonts configuration files
- **xfont50.tgz** - Contains X11's font server and fonts
- **xserv50.tgz** - Contains X11's X servers
- **xshare50.tgz** - Contains manpages, locale settings, includes, etc. for X

The `etc50.tgz` and `xetc50.tgz` sets are not installed as part of an upgrade, only as part of a complete install, so any customizations you make will not be lost. You will have to update your `/etc`, `/dev` and `/var` directories manually.

Why do I have to install X for my non-graphical application?

Even if you have no intention of running X, some third party packages require the libraries or other utilities in X to be installed on your system. These applications can sometimes be satisfied simply by installing just `xbase50.tgz`, the rest of X is not always needed. Many people resist installing X on their system without valid reason:

- By itself, installing X does not cause any program to execute on the system.
- By itself, installing X on a system does not change the risk of external security issues.
- If someone is already ON your system, they can most likely install whatever they wish, so the presence or absence of the X does not appreciably change the situation.
- The only parts of X that are running are the parts required by your application.
- The space required for X is relatively modest on modern hardware.

People sometimes waste a lot of time and effort trying to pick through `xbase50.tgz` and pull out just the files they need to install their application. This is not only pointless, but an effort that would have to be repeated for each upgrade cycle, which probably means you will not upgrade your system properly, creating REAL security problems. IF you need X, just install it. It won't hurt you any more than the application you are needing it for will.

I don't want to install the compilers

Ok, don't, but please don't tell yourself this is for "security reasons". By the time someone is far enough into your system that the presence or absence of the compiler matters, they are far enough in they can install a compiler themselves. However, the `compXX.tgz` file set is relatively big and has a lot of files in it, so it can take a while to install and upgrade, and on slow or small systems, this can matter.

If you do decide to not install the compiler, you will probably need another system to maintain and build updated software on. There are far more systems that have been compromised because of improper maintenance than there have been because a compiler was installed.

4.8 How should I partition my disk?

Obviously, the answer to this question varies tremendously based on your use of the system. OpenBSD can be installed in as little as 512M, but installing to that small of a device is something for advanced users. Until you have some experience, a 4G to 8G HD is recommended to start with.

Unlike many other OSs, OpenBSD encourages users to partition their disk into a number of partitions, rather than having just one or two big partitions. There are a number of reasons to partition your disk:

- **Security**: You can mark some filesystems as '`nosuid`', '`nodev`', '`noexec`', '`readonly`', etc. This is done for you by the install process, if you use the recommended partitions.
- **Stability**: A user, or a misbehaved program, can fill a filesystem with garbage if they have write permissions for it. Your critical programs, which hopefully run on a different filesystem, do not get interrupted.
- **Speed**: A filesystem which gets written to frequently may get somewhat fragmented. (Luckily, the ffs filesystem that OpenBSD uses is not prone to heavy fragmentation.)
- **Integrity**: If one filesystem is corrupted for some reason then your other filesystems are most likely still OK.
- **Size**: Many machines have limits on the area of a disk where the boot ROM can load the kernel from. In some cases, this limit may be very small (504M for an older 486), in other cases, a much larger limit (for example, 2G, 8G, or 128G on i386 systems). As the kernel can end up anywhere within the root partition, the entire root partition should be within this area. For more details, see this section. A good guideline might be to keep your `/` partition completely below 2G, unless you know your platform (and particular machine) can handle more (or less) than that.
- **Read-only**: You can mount partitions that you never or rarely need to write to as "Read-only" most of the time, which will eliminate the need

for `fsck(8)` after a crash or power interruption, and may help prevent unintended data alteration.

- **`fsck(8)`**: Very large partitions require more RAM to `fsck(8)`, and on small-memory systems, you can end up having to use swap, resulting in very long fsck times.

Given sufficient disk space, OpenBSD's installer will default to the following partitions:

- **`/` - root**: In addition to being where the other file systems are mounted, the root file system holds all the files needed to boot OpenBSD. This includes the kernel, plus the basic utilities in `/sbin` and `/bin`, the configuration files in /etc, and the device directory, `/dev` which are all used to bring up the system. The root file system can be as small as 60M, though 100M to 200M is easier for a machine that will last through many upgrade cycles. The '`a`' partition of your boot drive becomes your root partition automatically. SOME platforms place restrictions on the physical location on the disk (i.e., must be at start of disk) in order to boot.

- **`Swap`**: In addition to swap, this partition is also used for storing core dumps after system crashes, so it is suggested that the swap space (if set up at all) be bigger than the largest amount of RAM you are likely to ever install on the machine. Read more about this in FAQ 14, Swap.

- **`/tmp`**: This is a world-writeable directory used for (as the name implies!) temporary storage. Most systems can get by with very modest amounts of storage here, 50M is usually many times what you should ever need, though there are a few applications which can use much, much more. While this directory is world-writable, when it is a separate partition, OpenBSD defaults to mounting it nodev and nosuid, which minimizes how it can be used to abuse your system Note that this directory is cleared on reboot, and files left untouched over 24 hours are cleared nightly.

- **`/var`**: This directory and mount point is used for a LOT of things, and depending on your uses, may be a prime candidate to subdivide into more partitions. Some of the things that end up here (and potential additional mount points):

  `/var/log`: System logs.

  `/var/mail`: Incoming mail boxes.

  `/var/spool`: Outgoing mail (and other things)

  `/var/www`: OpenBSD's web server lives here.

  `/var/tmp`: This is a "persistent" temporary file directory, as files placed here are NOT purged on reboot. For example, `vi(1)` uses this directory for temporary storage, so if your computer crashes or is rebooted while you are editing a file, the files here can be used to recover your editing

session. Files left here over 24 hours though will be purged by the nightly cleanup scripts, `daily(8)`.

- `/var/crash`: If the system panics, it will attempt to save a core dump in the swap partition before rebooting. This core dump will then be saved to `/var/crash` upon reboot, so `/var` will need at least as much free space as the system has RAM for this to work automatically.
- **`/usr`**: This is where most of OpenBSD resides. Program binaries, libraries, documentation, manual pages, etc. are all located in the /usr directory. The files in this mount point are relatively unchanging – in many cases, you could easily mount the `/usr` partition read-only with no other system changes until your next upgrade or update.
- **`/usr/X11R6`**: This is where the X Window system resides. The X binaries, font files, libraries, etc. all are here. The only part of X not here is the configuration files.
- **`/usr/local`**: On a default OpenBSD installation, this mount point/directory is completely empty. It is used for locally installed binaries and libraries for local applications.
- **`/usr/src`**: This directory holds the basic system source files, excluding X and ports. This directory is empty by default, you have to load it as detailed in FAQ 5.
- **`/usr/obj`**: This directory is populated during the build process with the object and binary files. Having this directory a separate mount point allows it to be formatted rather than erased file by file, which can be significantly faster.
- **`/home`**: This is where user files go. Having this a separate partition makes it easy to completely reinstall your system; simply don't format this partition on reload.

Some additional thoughts on partitioning:

- For your first attempt at an experimentation system, one big `/` partition and swap may be easiest until you know how much space you need. By doing this you will be sacrificing some of the default security features of OpenBSD that require separate filesystems for `/`, `/tmp`, `/var`, `/usr` and `/home`. However, you probably should not be going into production with your first OpenBSD install.
- A system exposed to the Internet or other hostile forces should have a separate `/var` (and maybe even a separate `/var/log`) for logging.
- A `/home` partition can be nice. New version of the OS? Wipe and reload everything else, leave your `/home` partition untouched. Remember to save a copy of your configuration files, though!

- A separate partition for anything which may accumulate a large quantity of files that may need to be deleted can be faster to reformat and recreate than to delete. See the building by source FAQ for an example (`/usr/obj`).

- If you wish to rebuild your system from source for any reason, the source will be in `/usr/src`. If you don't make a separate partition for `/usr/src`, make sure /usr has sufficient space.

- A commonly forgotten fact: you do not have to allocate all space on a drive when you set the system up! Since you will now find it a challenge to buy a new drive smaller than 100G, it can make sense to leave a chunk of your drive unallocated. If you outgrow a partition, you can allocate a new partition from your unused space, duplicate your existing partition to the new partition, change `/etc/fstab` to point to the new partition, remount, you now have more space.

- If you make your partitions too close to the minimum size required, you will probably regret it later, when it is time to upgrade your system.

- If you make very large partitions, keep in mind that performing filesystem checks using `fsck(8)` requires about 1M of RAM per gigabyte of filesystem size, and may be very time-consuming or not even feasible on older, slower systems (please also refer to this section).

- If you permit users to write to `/var/www` (i.e., personal web pages), you might wish to put it on a separate partition, so you can use quotas to restrict the space they use, and if they fill the partition, no other parts of your system will be impacted.

- You may also want to create an `/altroot` partition, as described in `daily(8)`. This can make a daily copy of your `/` partition, giving you an extra copy of your kernel and /etc configuration files should something happen to your root partition. Obviously, the `/altroot` partition needs to be at least as big as `/`. If you have a second drive and have something else duplicating the rest of your disk, either software `softraid(4)` or a periodic copy using `dump(8)`/`restore(8)`, this disk can be bootable after the removal of the primary disk.

- Compiling some ports from source can take huge amounts of space on your `/usr` and `/tmp` partitions. This is another reason we suggest using pre-compiled packages instead.

- At least some editors use `/var/tmp` for scratch space, and this often needs to be as big or bigger than the largest file you edit. If you plan on editing 500M files, your `/var` or `/var/tmp` partition will need to be much larger than your might have planned on.

4.9 Multibooting OpenBSD (amd64, i386)

Multibooting is having several operating systems on one computer, and some means of selecting which OS is to boot. It is *not* a trivial task! If you don't understand what you are doing, you may end up deleting large amounts of data from your computer. New OpenBSD users are *strongly* encouraged to start with a blank hard drive on a dedicated machine, and then practice your desired configuration on a non-production system before attempting a multiboot configuration on a production machine. FAQ 14 has more information about the OpenBSD boot process.

Preferably use one of the four *primary* MBR partitions for booting OpenBSD (i.e., extended partitions may not work).

Note that Windows 7 and Vista can resize their system partitions: go to the Control Panel, search for "partition", and enter the corresponding system tool. Right click on a partition, and you will notice you can shrink it. Its main limitation is that the Windows Exchange File can't be moved, so if you need more space, you may have to move/disable it.

Here are several options to multibooting:

Setting active partitions

This is probably the most overlooked, and yet, sometimes the best solution for multibooting. Simply set the active partition in whatever OS you are currently using to be the one you want to boot by default when you next boot. Virtually every OS offers a program to do this; OpenBSD's is `fdisk(8)`, similar named programs are in Windows 9x and DOS, and many other operating systems. This can be highly desirable for OSs or systems which take a long time to shut down and reboot – you can set it and start the reboot process, then walk away, grab a cup of coffee, and come back to the system booted the way you want it – no waiting for the Magic Moment to select the next OS.

Boot floppy

If you have a system that is used to boot OpenBSD infrequently (or don't wish other users of the computer to note anything has changed), consider using a boot floppy. Simply use one of the standard OpenBSD install floppies, and create an `/etc/boot.conf` file (yes, you will also have to create an texttt/etc directory on the floppy) with the contents:

```
boot hd0a:/bsd
```

to cause the system to boot from hard drive 0, OpenBSD partition 'a', kernel file `/bsd`. Note you can also boot from other drives with a line like: "`boot hd2a:/bsd`" to boot off the third hard drive on your system. To boot from OpenBSD, slip your floppy in, reboot. To boot from the other OS, eject the floppy, reboot. (You can, of course, use this floppy to make a bootable CD, too.)

The `boot(8)` program is loaded from the floppy, looks for and reads `/etc/boot.conf`. The "`boot hd0a:/bsd`" line instructs `boot(8)` where to load the kernel from – in this case, the first HD the BIOS sees. Keep in mind, only a small file (`/boot`)

is loaded from the floppy – the system loads the entire kernel off the hard disk, so this only adds about five seconds to the boot process.

Windows NT/2000/XP NTLDR

To multiboot OpenBSD and Windows NT/2000/XP, you can use NTLDR, the boot loader that NT uses. To multi-boot with NT, you need a copy of your OpenBSD Partition Boot Record (PBR). After running installboot, you can copy it to a file using `dd(1)`, following a process similar to:

```
# dd if=/dev/rsd0a of=openbsd.pbr bs=512 count=1
```

Note: this is a really good time to remind you that blindly typing commands in you don't understand is a really bad idea. This line will not work directly on most computers. It is left to the reader to adapt it to their machine.

Now boot NT and put `openbsd.pbr` in C:. Add a line like this to the end of `C:\BOOT.INI`:

```
c:\openbsd.pbr="OpenBSD"
```

When you reboot, you should be able to select OpenBSD from the NT loader menu. There is much more information available about NTLDR at the NTLDR Hacking Guide.

On Windows XP you can also edit the boot information using the GUI; see the XP Boot.ini HOWTO.

Programs that do much of this for you are available, for example, BootPart. This program can be run from Windows NT/2000/XP, and will fetch the OpenBSD PBR, place it on your NT/2000/XP partition, and will add it to `C:\BOOT.INI`.

Note: The Windows NT/2000/XP boot loader is only capable of booting OSs from the primary hard drive. You can not use it to load OpenBSD from the second drive on a system.

Windows Vista

With Vista, Microsoft dropped NTLDR support in favor of their newer Boot Configuration Data (BCD) store used for controlling the boot environment. Since `BOOT.INI` is no longer available for customization, a command-line utility, bcdedit, takes its place.

Once OpenBSD's PBR is copied to Vista's system partition, the following three commands are required to select and boot OpenBSD when the system is restarted:

```
C:\Windows\system32> bcdedit /create /d "OpenBSD/i386" /application bootsector
The entry {05a763ce-d81b-11db-b3ec-000000000000} was successfully created.

C:\Windows\System32>
```

The GUID returned here, `05a763ce-d81b-11db-b3ec-000000000000`, is shown for illustrative reasons. Take note of the GUID displayed when you run this command as this value will need to be copied into the following commands. Simply copying the GUID shown above will not work.

The following two commands are also required:

```
C:\Windows\System32> bcdedit /set 05a763ce-d81b-11db-b3ec-000000000000 device boot
The operation completed successfully.

C:\Windows\System32> bcdedit /set 05a763ce-d81b-11db-b3ec-000000000000 path \openbsd.pbr
The operation completed successfully.

C:\Windows\System32>
```

This must be run in a shell with administrative privileges. Once you've located cmd.exe, right click to be able to select "run as administrator".

Note the absolute pathname of the imported PBR file. Do not add a drive letter as it is assumed that the file is placed in the system partition. bcdedit will not complain about an explicit drive specification, but the boot manager will later balk claiming that it cannot resolve the designated pathname.

Upon rebooting, Vista will be listed first in the boot manager ultimately followed by OpenBSD. Selecting either entry will boot the corresponding operating system.

If nothing happens, look around in the control panel for boot information. Most likely, your Windows boot is set up with no delay, so you don't see the boot menu. You can also use this to boot OpenBSD by default.

For more information, consult `bcdedit`'s help by issuing:

```
C:\Windows\System32> bcdedit /?
```

or by searching Microsoft's documentation and Website. A good introduction can be found in this TechNet Frequently Asked Questions article.

For those who find manual configuration daunting, EasyBCD provides a GUI alternative.

Windows 7

Microsoft has enhanced BCD since releasing Vista to allow multiple versions of Windows to be booted through `bcdedit`. Because of this greater control, five commands are required to configure a multiboot environment with OpenBSD.

After copying OpenBSD's PBR into Windows 7's system partition, issue the following command to initialize the needed registry hive:

```
C:\Windows\System32> bcdedit /create /d "OpenBSD/i386" /application bootsector
The entry 0154a872-3d41-11de-bd67-a7060316bbb1 was successfully created.

C:\Windows\System32>
```

As admonished before, the {`0154a872-3d41-11de-bd67-a7060316bbb1`} GUID is system-dependent. Note the value you receive when executing, and copy it into the following commands:

```
C:\Windows\System32> bcdedit /set 0154a872-3d41-11de-bd67-a7060316bbb1 device boot
The operation completed successfully.
```

```
C:\Windows\System32> bcdedit /set 0154a872-3d41-11de-bd67-a7060316bbb1 path \openb
The operation completed successfully.

C:\Windows\System32> bcdedit /set 0154a872-3d41-11de-bd67-a7060316bbb1 device part
The operation completed successfully.

C:\Windows\System32> bcdedit /displayorder 0154a872-3d41-11de-bd67-7060316bbb1 /ad
The operation completed successfully.

C:\Windows\System32>
```

Other boot loaders

Some other bootloaders OpenBSD users have used successfully include GAG, The Ranish Partition Manager, rEFIt, and GRUB.

OpenBSD and Linux (i386)

Please refer to `INSTALL.linux`, which gives in depth instructions on getting OpenBSD working with Linux.

Time zone issues

OpenBSD expects the computer's real-time clock to be set to UTC (Universal Coordinated Time). Some other OSs expect the real-time clock to be set to local time. Obviously, this can create a bit of a problem if you are using both OSs on the same computer. One or the other is most likely going to have to be adapted. More info on doing this is in FAQ 8 - Why is my clock off by several hours?

4.10 Sending your dmesg to dmesg@openbsd.org after the install

Just to remind people, it's important for the OpenBSD developers to keep track of what hardware works, and what hardware doesn't work perfectly, including the hardware sensors that are found in machines.

A quote from `/usr/src/etc/root/root.mail`

```
    If you wish to ensure that OpenBSD runs better on your machines, please do us
a favor (after you have your mail system configured!) and type something like:
 # (dmesg; sysctl hw.sensors) | \
  mail -s "Sony VAIO 505R laptop, apm works OK" dmesg@openbsd.org
so that we can see what kinds of configurations people are running.  As shown,
including a bit of information about your machine in the subject or the body
can help us even further.  We will use this information to improve device driver
support in future releases.  (Please do this using the supplied GENERIC kernel,
not for a custom compiled kernel, unless you're unable to boot the GENERIC
kernel.  If you have a multi-processor machine, dmesg results of both GENERIC.MP
and GENERIC kernels are appreciated.)  The device driver information we get from
this helps us fix existing drivers. Thank you!
```

Make sure you send email from an account that is able to also receive email so developers can contact you if they have something they want you to test or change in order to get your setup working. It's not important at all to send the email from the same machine that is running OpenBSD, so if that machine is unable to receive email, just

```
$ (dmesg; sysctl hw.sensors) | mail your-account@yourmail.dom
```

and then forward that message to

```
dmesg@openbsd.org
```

where `your-account@yourmail.dom` is your regular email account.

NOTES

- Please send only GENERIC kernel dmesgs. Custom kernels that have device drivers removed are not helpful.
- If you have a supported multiprocessor system and normally run the GENERIC.MP kernel, it is helpful to developers to see the dmesg output of both the GENERIC kernel and the GENERIC.MP kernel, so please send both of them in separate emails.
- The dmesgs are received on a computer using the spamd spam rejection system. This may cause your dmesg to not be accepted by the mail servers for a period of time. Be patient, after half an hour to an hour or so, it will get through.

The method above is very easy, but if you have chosen not to configure mail on your OpenBSD system, you should still send your dmesg to the developers. Save your dmesg output to a text file.

```
$ (dmesg; sysctl hw.sensors) > ~/dmesg.txt
```

Then transfer this file (using FTP/scp/floppydisk/carrier-pigeon/...) to the system you normally use for email. Since the dmesg output you send in is processed automatically, be sure to check the following when using alternate email clients/systems:

- Configure your email client to send messages as plain text; do not use HTML-formatted email.
- Turn off any forced line break feature. Many email clients are configured to insert line breaks at 72 columns (the norm for mailing lists).
- Make sure your email client does not reformat messages into "text-flow" nonsense.
- Do not send the dmesg output as file attachment. Put the dmesg output into the body of the message.

4.11 Adding a file set after install

"Oh no! I forgot to add a file set when I did the install!"

Sometimes, you realize you really DID need `comp50.tgz` (or any other system component) after all, but you didn't realize this at the time you installed your system. Good news: There are two easy ways to add file sets after the initial install:

Using the upgrade process

Simply boot your install media (CD-ROM or Floppy), and choose Upgrade (rather than Install). When you get to the lists of file sets to install, choose the sets you neglected to install first time around, select your source, and let it install them for you.

Using `tar(1)`

The install file sets are simply compressed tar files, and you can expand them manually from the root of the filesystem:

```
# cd /
# tar xzvphf comp50.tgz
```

Do NOT forget the 'p' option in the above command in order to restore the file permissions properly!

One common mistake is to think you can use `pkg_add(1)` to add missing file sets. This does not work. `pkg_add(1)` is the package management tool to install third party software. It handles package files, not generic tar files like the install sets.

If you are installing the `xbase` file set on your system for the first time using tar(1) and without rebooting, the shared library cache must be updated after the installation using `ldconfig(8)`. To add all the X libraries to the cache:

```
# ldconfig -m /usr/X11R6/lib
```

Alternatively, you can just reboot your system, and this will be done automatically by the rc(8) startup script.

4.12 What is '`bsd.rd`'?

`bsd.rd` is a "RAM Disk" kernel. This file can be very useful; many developers are careful to keep it on the root of their system at all times.

Calling it a "RAM Disk kernel" describes the root filesystem of the kernel – rather than being a physical drive, the utilities available after the boot of `bsd.rd` are stored in the kernel, and are run from a RAM-based filesystem. `bsd.rd` also includes a healthy set of utilities to allow you to do system maintenance and installation.

On some platforms, `bsd.rd` is actually the preferred installation technique – you place this kernel on an existing filesystem, boot it, and run the install from it. On most platforms, if you have a running older version of OpenBSD, you

can FTP a new version of `bsd.rd`, reboot from it, and install a new version of OpenBSD without using any removable media at all.

Here is an example of booting `bsd.rd` on an i386 system:

```
  Using Drive: 0 Partition: 3
  reading boot.....
  probing: pc0 com0 com1 apm mem[639k 255M a20=on]
  disk: fd0 hd0+
  >> OpenBSD/i386 BOOT 3.17
  boot> boot hd0a:/bsd.rd
. . . normal boot to install . . .
```

As indicated, you will be brought to the install program, but you can also drop to the shell to do maintenance on your system. The general rule on booting `bsd.rd` is to change your boot kernel from `/bsd` to `bsd.rd` through whatever means used on your platform.

4.13 Common installation problems

4.13.1 My Compaq only recognizes 16M RAM

Some Compaq systems have an issue where the full system RAM is not detected by the OpenBSD second stage boot loader properly, and only 16M may be detected and used by OpenBSD. This can be corrected either by creating/editing `/etc/boot.conf` file, or by entering commands at the "boot>" prompt before OpenBSD loads. If you had a machine with 64M RAM, but OpenBSD was only detecting the first 16M, the command you would use would be:

```
machine mem +0x3000000@0x1000000
```

to add 48M (`0x3000000`) after the first 16M (`0x1000000`). Typically, if you had a machine with this problem, you would enter the above command first at the install floppy/CD-ROM's boot>prompt, load the system, reboot, and create an `/etc/boot.conf` file with the above line in it so all future bootings will recognize all available RAM.

It has also been reported that a ROM update will fix this on *some* systems.

4.13.2 My i386 won't boot after install

Your install seemed to go fine, but on first boot, you see no sign of OpenBSD attempting to boot. There are a few common reasons for this problem:

- **No partition was flagged active in *fdisk(8)***. To fix this, reboot the machine using the boot floppy or media, and "flag" a partition as "active" (bootable). See here and here.

- **No valid boot loader was ever put on the disk**. If you hit "Enter" or answer "w" to the "Use (W)hole disk or (E)dit the MBR?" question during the install, or use the "reinit" option of fdisk(8), the OpenBSD boot record is installed on the Master Boot Record of the disk; otherwise, the existing master boot code is untouched. This will be a problem if no other boot record existed. One solution is to boot the install media again, drop to the shell and invoke fdisk(8) to update the MBR code from the command line:

  ```
  # fdisk -u wd0
  ```

  Note: the "update" option within the interactive ("`-e`") mode of fdisk will not write the signature bytes required to make the disk bootable.

- **In some rare occasions, something may go wrong with the second stage boot loader install**. Reinstalling the second stage boot loader is discussed here.

4.13.3 My (older, slower) machine booted, but hung at the `ssh-keygen` steps

It is very likely your machine is running fine, just taking a while to do the ssh key generation process. A SPARCStation2 or a Macintosh Quadra may take *several hours* to complete the `ssh-keygen(1)` steps. Just let it finish; it is only done once per install.

Users of very slow machines may wish to generate their keys on another computer, place them in a `site50.tgz` file, and install them with the rest of the file sets.

4.13.4 I got the message "Failed to change directory" when doing an install

When doing an FTP install of a snapshot during the -beta stage of the OpenBSD development cycle, you may see this:

```
Display the list of known ftp servers? [no] yes
Getting the list from 192.128.5.191 (ftp.openbsd.org)... FAILED
Failed to change directory.
Server IP address or hostname?
```

This is normal and expected behavior during this pre-release part of the cycle. The install program looks for the FTP list on the primary FTP server in a directory that won't be available until the release date, so you get the above message.

Simply use the FTP mirror list to find your favorite FTP mirror, and manually enter its name when prompted.

Note: You should not see this if you are installing *-release* or from CD-ROM.

4.13.5 My fdisk partition table is trashed or blank!

Occasionally, a user will find a system will work, but when doing an `fdisk wd0`, they see a completely blank (or sometimes, garbage) partition table. This is usually caused by having created a partition in `fdisk(8)` which had an offset of zero sectors, rather than the one track offset it should have (note: this is assuming the i386 or amd64 platform. Other platforms have different offset requirements, some need NO offset). The system then boots using the PBR, not using the MBR.

While this configuration can work, it can be a maintenance problem and should be fixed. To fix this, the disk's file systems must generally be recreated from scratch (though if you REALLY know what you are doing, you may be able to recreate just your disklabel and MBR, and only lose and have to rebuild the first OpenBSD partition on the disk).

4.13.6 I have no floppy or CD-ROM on my machine

Some computers people might want to run OpenBSD on lack any obvious way to install OpenBSD, having no floppy or CD-ROM drive. Either the machine was designed without it (for example, many laptops and "flash" based machines, like Soekris and ALIX systems), or the boot devices have failed or been removed, and would be difficult to replace. Here are some tips and techniques you can use to get OpenBSD installed on these systems.

- Network boot, using PXE (i386 or amd64) or `diskless(8)` (other platforms).
- External USB CD-ROM or USB floppy, if your machine can boot from one.
- USB Flash disk or hard disk, again if your computer can boot from a USB device. Prepare the device on another computer as described in FAQ 14. Boot from it, but chose the bsd.rd kernel, then install as normal. You could also have the file sets pre-loaded on the flash media, as well.
- Worst case, if none of the above is suitable, you can usually pull the disk out of the target system, use suitable adapters to install it in a "normal" computer, install OpenBSD, then replace the disk back in the target system. OpenBSD will then boot nicely in the target machine, though you will very possibly have to adjust the network configuration. You may also have to adjust `/etc/fstab` if you (for example) did your install using a USB->IDE/SATA adapter or your target or install machine uses ahci(4) and the other does not. IDE and some SATA disks will normally be recognized as `wd(4)` devices, but if attached to a USB adapter, will come up as `sd(4)`. A SATA disk attached to a `pciide(4)` interface will come up as `wd(4)`, but attached to an `ahci(4)` interface, will come up as a `sd(4)` device. However, once you correct the `/etc/fstab` file, the system will generally boot just fine.

In all cases, remember that the machine had an OS installed on it before, and it was usually intended that the OS could be reloaded in the field. How this was originally intended to be done will often provide you a good idea how you can install OpenBSD now.

4.13.7 I got an SHA256 mismatch during install!

Checksums are embedded in the install kernels for the file sets that are used for the system install.

Actual *-release* file sets should all match their stored checksums.

At times, snapshots may not have proper checksums stored with the install kernels. This will happen for various reasons on the building side, and is not reason to panic for development snapshots. If you are concerned about this, wait for the next snapshot.

4.14 Customizing the install process

`siteXX.tgz` file

The OpenBSD install/upgrade scripts allow the selection of a user-created set called "`siteXX.tgz`", where XX is the release version (e.g. 50). The `siteXX.tgz` file set is, like the other file sets, a `gzip(1)` compressed `tar(1)` archive rooted in '/' and is un-tarred like the other sets with the options `xzphf`. This set will be installed last, after all other file sets.

This file set allows the user to add to and/or override the files installed in the 'normal' sets and thus customize the installation or upgrade.

You can also create and use hostname-specific install sets, which are named `siteXX-`<`hostname`>.tgz, for example, "`site50-puffy.tgz`". This allows easy per-host customized installations, upgrades, or disaster recovery.

Some example uses of a `siteXX.tgz` file:

- Create a `siteXX.tgz` file that contains all the file changes you made since first installing OpenBSD. Then, if you have to re-create the system you simply select `siteXX.tgz` during the re-install and all of your changes are replicated on the new system.
- Create a series of machine specific directories that each contain a `siteXX.tgz` file that contains files specific to those machine types. Installation of machines (e.g. boxes with different graphics cards) of a particular category can be completed by selecting the appropriate `siteXX.tgz` file.
- Put the files you routinely customize in a same or similar way in a `siteXX.tgz` file – `/etc/skel` files, `/etc/pf.conf`, `/var/www/conf/httpd.conf`, `/etc/rc.conf.local`, etc.

`install.site/upgrade.site` scripts

As the last step in the install/upgrade process, the scripts look in the root directory of the newly installed/upgraded system for `install.site` or

`upgrade.site`, as appropriate to the current process, and runs this script in an environment chrooted to the installed/upgraded system's root. Remember, the upgrade is done from a booted file system, so your target file system is actually mounted on `/mnt`. However, because of the chroot, your script can be written as if it is running in the "normal" root of your file system. Since this script is run after all the files are installed, you have much of the functionality of the full system when your script runs. Keep in mind that you are running a minimal kernel, not all features are available, and due to space constraints, things that work today may not work in a future release.

Note that the `install.site` script would have to be in a `siteXX.tgz` file, while the `upgrade.site` script could be put in the root directory before the upgrade, or could be put in a `siteXX.tgz` file.

The scripts can be used to do many things:

- Remove files that are installed/upgraded that you don't want present on the system.
- Remove/upgrade/install the packages you want on the installed system (may not work for all packages!).
- Do an immediate backup/archive of the new system before you expose it to the rest of the world.
- Use `rdate(8)` to set the system time.
- Have a set of arbitrary commands be run after the first boot. This will happen if install.site is used to append any such commands to an rc.firsttime(8) file (appending to this file is neccessary since the installer itself may write to this file). At boot time, `rc.firsttime` is executed once then deleted.

The combination of `siteXX.tgz` and `install.site`/`upgrade.site` files is intended to give users broad customization capabilities without having to build their own custom install sets.

Note: if you will be doing your install from an http server, you will need to add your `site*.tgz` file(s) to the file `index.txt` in the source directory in order for them to be listed as an option at install time. This is not needed for FTP or other installs.

4.15 How can I install a number of similar systems?

Here are some tools you can use when you have to deploy a number of similar OpenBSD systems.

`siteXX.tgz` and `install`/`upgrade.site` files

See the above article.

Restore from `dump(8)`

On most platforms, the boot media includes the `restore(8)` program, which can be used to restore a backup made by `dump(8)`. Thus, you could boot from a floppy, CD, or `bsd.rd` file, then `fdisk`, `disklabel`, and `restore` the desired configuration from tape or other media, and install the boot blocks. More details here.

Disk imaging

Unfortunately, there are no known disk imaging packages which are FFS-aware and can make an image containing only the active file space. Most of the major disk imaging solutions will treat an OpenBSD partition as a "generic" partition, and can make an image of the whole disk. This often accomplishes your goal, but usually with huge amounts of wasted space – an empty, 10G `/home` partition will require 10G of space in the image, even if there isn't a single file in it. While you can typically install a drive image to a larger drive, you would not be able to directly use the extra space, and you would not be able to install an image to a smaller drive.

If this is an acceptable situation, you may find the `dd` command will do what you need, allowing you to copy one disk to another, sector-for-sector. This would provide the same functionality as commercial programs without the cost.

4.16 How can I get a dmesg(8) to report an install problem?

When reporting a problem, it is critical to include the complete system `dmesg(8)`. However, often when you need to do this, it is because the system is working improperly or won't install so you may not have disk, network, or other resources you need to get the dmesg to the appropriate mail list. There are other ways, however:

- **Floppy disk**: The boot disks and CD-ROM have enough tools to let you record your dmesg to an MSDOS floppy disk for reading on another machine. Place an MSDOS formatted floppy in your disk drive and execute the following commands:

  ```
  mount -t msdos /dev/fd0a /mnt
  dmesg >/mnt/dmesg.txt
  umount /mnt
  ```

  If you have another OpenBSD system, you can also write it to an OpenBSD compatible floppy – often, the boot floppy has enough room on it to hold the `dmesg`. In that case, leave off the "`-t msdos`" above.

- **Serial Console**: Using a serial console and capturing the output on another computer is often the best way to capture diagnostic information - particularly if the computer panics immediately after boot. As well as a

second computer, you will need a suitable serial cable (often a null-modem cable), and a terminal emulator program that can capture screen output to file. General information on setting up a serial console is provided elsewhere in the FAQ; in order to capture a log of the install, the following commands are usually sufficient.

i386

At the boot loader prompt, enter

```
boot> set tty com0
```

This will tell OpenBSD to use the first serial port (often called COM1 or COMA in PC documentation) as a serial console. The default baud rate is 9600.

Sparc/Sparc64

These machines will automatically use a serial console if started without a keyboard present. If you have a keyboard and monitor attached, you can still force the system to use a serial console with the following invocation at the `ok` prompt.

```
ok setenv input-device ttya
ok setenv output-device ttya
ok reset
```

Chapter 5

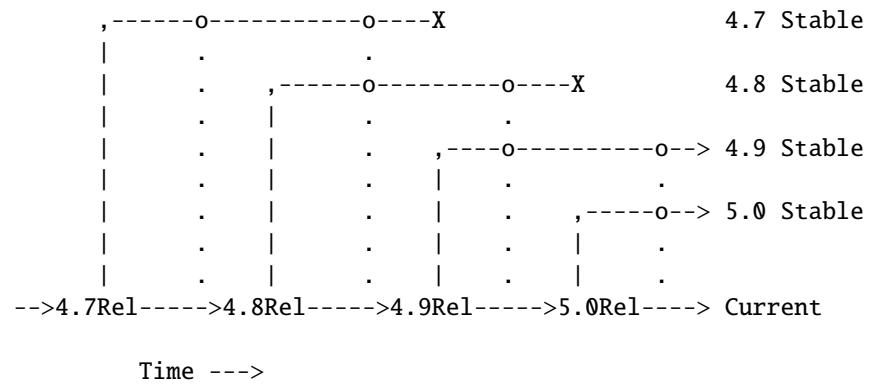
Building the System from Source

5.1 OpenBSD's Flavors

There are three "flavors" of OpenBSD:

- **-release**: The version of OpenBSD shipped every six months on CD.
- **-stable**: Release, plus patches considered critical to security and reliability.
- **-current**: Where new development work is presently being done, and eventually, it will turn into the next release.

Graphically, the development of these flavors looks something like this:



-Current is where active development work is done, and eventually, it will turn into the next *-release* of OpenBSD. Every six months, when a new version of OpenBSD is released, *-current* is tagged, and becomes *-release*: a frozen point in the history of the source tree. Each *-release* is never changed; it is what is on the CDs and FTP servers.

-Stable is based on *-release*, and is a branch from the main development path of OpenBSD. When very important fixes are made to *-current*, they are "back ported" (merged) into the *-stable* branches; because of this, *-stable* is also known as the "*patch branch*." In the above illustration, the vertical dotted lines denote bug fixes being incorporated into the *-stable* branches. You will also note that in the above example, the *4.7-stable* branch came to an end with *4.9-release*, and the *4.8-stable* branch came to an end with *5.0-release* – old releases are typically supported up to two releases back. It takes resources and time to support older versions, while we might like to provide ongoing support for old releases, we would rather focus on new features. The *-stable* branch is, by design, very easy to build from *-release* of the same version (i.e., going from *5.0-release* to *5.0-stable*).

The *-stable* branch is *-release* plus patches found on the errata page. The operation of *-stable* is the same as the *-release* it is based on. If the man pages have to change, it probably won't go into *-stable*. In other words, new device support and new features will NOT be added to *-stable*.

It is worth pointing out that the name "*-stable*" is not intended to imply that *-current* is unreliable or less robust in production. Rather, the APIs (how the programs talk to the OS) and features of *-current* are changing and evolving, whereas the operation and APIs of *-stable* are the same as the release it is based on, so you shouldn't have to relearn features of your system or change any configuration files, or have any problem adding additional applications to your system.

In fact, as our hope is to continually improve OpenBSD, the goal is that *-current* should be more reliable, more secure, and of course, have greater features than *-stable*. Put bluntly, the "best" version of OpenBSD is *-current*.

Most users should be running either *-stable* or *-release*. That being said, many people do run *-current* on production systems, and it is important that people do so to identify bugs and test new features. However, if you don't know how to properly describe, diagnose and deal with a problem, don't tell yourself (or anyone else) that you are "helping the project" by running *-current*. "It didn't work!" is not a useful bug report. "The recent changes to the pciide driver broke compatibility with my Slugchip-based IDE interface, dmesg of working and broken systems follow..." might be a useful report.

There are times when "normal" users may wish to live on the cutting edge and run *-current*. The most common reason is that the user has a device which is not supported by *-release* (and thus, not *-stable*), or wishes to use a new feature of *-current*. In this case, the choice may be either *-current* or not using the device, and *-current* may be the lesser evil. However, one should not expect hand-holding from the developers.

Snapshots

Between formal releases of OpenBSD, *snapshots* are made available through the FTP sites. As the name implies, these are builds of whatever code is in the tree at the instant the builder grabbed a copy of the code for that particular platform. Remember, on some platforms, it may be DAYS before the snapshot build is completed and put out for distribution. There is no promise that the snapshots are completely functional, or even install. Often, a change that needs to be tested may trigger snapshot creation. Some platforms have snapshots built on an almost daily basis, others will be much less frequent. If you desire to run *-current*, a recent snapshot is often all you need, and upgrading to a snapshot is a required starting point before attempting to build *-current* from source.

It is sometimes asked if there is any way to get a copy of exactly the code used to build a snapshot. The answer is no. First, there is no significant benefit to this. Second, the snapshots are built as desired, as time permits, and as resources become available. On fast platforms, several snapshots may be released in one day. On slower platforms, it may take a week or more to build a snapshot. Providing tags or markers in the source tree for each snapshot would be quite impractical. Third, snapshots often contain experimental code that isn't yet committed to the tree.

Upgrade vs. Update

You will often see references to "upgrading" and "updating" OpenBSD installs. Even though these words have similar meanings, they are used slightly differently in OpenBSD.

Upgrading is the process of installing a newer version of OpenBSD, with new functionality. For example, going from v4.9 to v5.0, or going from the June 12th snapshot to the June 20th snapshot. When upgrading, you will typically have to consult either Following -current or the Upgrade guide (when changing releases) to make the changes required to run the upgraded version of OpenBSD.

Updating is the process of applying patches to a system to improve the operation WITHOUT changing the basic functionality or binary compatibility. This is typically done by following the source patching process or by following the stable process. When you "update" your system, it goes from a *-release* to a *-stable* (or patched *-release*) of the same release version, for example, 5.0-release to 5.0-stable. You may then later update it to a newer *-stable* of the same release version. The update process is typically very painless, as no `/etc` files or other system configurations need to be changed.

So, you may install a system (for example, 4.8-release) from CD, then update it to 4.8-stable a few times, then upgrade it to 4.9-release from CD, and update that a few times before upgrading it again to the 5.0-release.

Keeping Things in Sync

It is important to understand that OpenBSD is an Operating System, intended to be taken as a whole, not a kernel with a bunch of utilities stuck on. You must make sure your kernel, "userland" (the supporting utilities and files) and ports tree are all in sync, or unpleasant things will happen. Said another way (because people just keep making the error), you can not run brand new `ports` on a month old system, or rebuild a kernel from *-current* source and expect it to work properly with a *-release* userland. Yes, this does mean you need to upgrade your system if you want to run a new program which was added to the ports tree today. Sorry, but again, OpenBSD has limited resources available.

One should also understand that the upgrade process is supported in **only one direction: from older to newer**, and from *-stable* to *-current*. You can not run *5.0-current* (or a snapshot), then decide you are living too dangerously, and step back to *5.0-stable*. You are on your own if you choose any path other than the supported option of reloading your system from scratch, do not expect assistance from the OpenBSD development team.

Yes, this does mean you should think before committing yourself to using *-current*.

5.2 Why do I need to compile the system from source?

Actually, you very possibly do not.

Some reasons why NOT to build from source:

- Compiling your own system as a way of upgrading it is not supported.
- You will NOT get better system performance by compiling your own system.
- Changing compiler options is more likely to break your system than to improve it.

Some reasons why you might actually wish or need to build from source:

- Test or develop new features.
- Compiling the system puts a lot of stress on the computer, it can be a way to make sure the system you just put together or acquired is actually in pretty good operational condition.
- You wish to follow the stable branch.
- You wish to make a highly customized version of OpenBSD for some special application.

The OpenBSD team puts out new snapshots based on *-current* code on a very regular basis for all platforms. It is likely this will be all you need for running -current.

The most common reason to build from source is to follow the *-stable* branch, where building from source is the only supported option.

If you are compiling *-current* from source, it is HIGHLY recommended that you only do so from a machine which you have full console access to. There will be times in the development process where the mismatch between your new kernel and your old userland may render the system inaccessible via network. This is not an issue when properly building *-stable*.

5.3 Building OpenBSD from source

5.3.1 Overview of the building process

Building OpenBSD from source involves a number of steps:

- Upgrading to the closest available binary.
- Fetching the appropriate source code.
- Building the new kernel and booting from it.

- Building "userland" ("make build").

There are a couple additional steps that some users may wish to perform, depending on the purpose of the build and if X is installed:

- Building a "release".
- Building X.

5.3.2 Install or Upgrade to closest available binary

The first step in building from source is to make sure you have the closest available binary installed. Use this table to look at where you are, where you wish to go, and what binary you should start with:

You are at	Goal	Binary upgrade to	then ...
Old -release	New release	Newest release	Done!
-release	-stable	Newest release	Fetch & build *-stable*
Old -stable	-stable	Newest release	Fetch & build *-stable*
-release	-current	Latest Snapshot	(optional) Fetch & build *-current*
Old -current	-current	Latest Snapshot	(optional) Fetch & build *-current*

It is recommended that you install the binary by using the "Upgrade" option of the install media. If that is not possible, you can also unpack the binaries as described here. Regardless, you must do the entire upgrade process, including creating any users or other `/etc` directory changes needed.

5.3.3 Fetching the appropriate source code

OpenBSD source is managed using the CVS version control system, and cvs(1) is used to pull a copy of the desired source to your local machine for compilation. This can be done by using an AnonCVS server (a machine holding a publicly accessible copy of the entire CVS repository used by the OpenBSD project) or from a local CVS repository you maintain using the CVSup, or CVSync programs available as packages. If you have multiple machines you wish to maintain source code trees on, you may find it worth having a local CVS repository, created and maintained using CVSync or rsync.

After deciding which AnonCVS server you wish to use, you must "checkout" the source tree, after that, you then maintain the tree by running "updates", to pull updated files to your local tree.

The CVS(1) command has many options, some of them are **required** to checkout and update a useful tree. Other commands can cause a broken tree. Following and understanding directions here is important.

Following *-current*

In this case, we will assume we are using a public AnonCVS server, **anoncvs@anoncvs.example.org:/cvs**. We will also assume you are using sh(1) as

your command shell, if you are using a different shell, you will have to adjust some of these commands.

To checkout a *-current* CVS src tree, you can use the following:

```
# cd /usr
# export CVSROOT=anoncvs@anoncvs.example.org:/cvs
# cvs -d$CVSROOT checkout -P src
```

Once you have a tree, you can update it at a later time:

```
# cd /usr/src
# export CVSROOT=anoncvs@anoncvs.example.org:/cvs
# cvs -d$CVSROOT up -Pd
```

Following *-stable*

If you wish to check out an alternative "branch" of the tree, such as the *-stable* branch, you will use the "`-r`" modifier to your checkout:

```
# cd /usr
# export CVSROOT=anoncvs@anoncvs.example.org:/cvs
# cvs -d$CVSROOT checkout -rOPENBSD_5_0 -P src
```

This will pull the src files from the `OPENBSD_5_0` branch, which is also known as the "Patch branch" or "*-stable*". You would update the code similarly:

```
# cd /usr/src
# export CVSROOT=anoncvs@anoncvs.example.org:/cvs
# cvs -d$CVSROOT up -rOPENBSD_5_0 -Pd
```

Actually, CVS is nice enough to stick a "Tag" in the checked out file system, so you don't have to remember the "`-rOPENBSD_5_0`" part of the command line, it will remember this until you explicitly clear them or set a new tag by using the "`-A`" option to "`update`". However, it is probably better to provide too much info in your CVS command lines than too little.

While only the "`src`" tree has been shown so far, you will do the same steps for "`xenocara`" and "`ports`". As all parts of OpenBSD must be kept in sync, all trees you use should be checked out and updated at the same time. You can combine the checkouts into one line (*-stable* shown):

```
# export CVSROOT=anoncvs@anoncvs.example.org:/cvs
# cd /usr
# cvs -d$CVSROOT checkout -rOPENBSD_5_0 -P src ports xenocara
```

However, updates must be done directory-by-directory.

At this point, whether you followed -stable or *-current* you should have a usable source tree. Be very careful which tree you grab – it is easy to try to compile *-current* when aiming for *-stable*.

Pre-loading the tree: src.tar.gz, sys.tar.gz

While you can download the entire source tree from an AnonCVS server, you can often save a lot of time and bandwidth by "pre-loading" your source tree with the source files from either the OpenBSD CD or from an FTP server. This is particularly true if you are running *-stable*, as relatively few files change between this version and the *-release* it is based on.

To extract the source tree from the CD to `/usr/src` (assuming the CD is mounted on `/mnt`):

```
# cd /usr/src; tar xzf /mnt/src.tar.gz
# cd /usr; tar xzf /mnt/xenocara.tar.gz
# cd /usr; tar xzf /mnt/ports.tar.gz
```

The source files available for download from the FTP servers are separated into two files to minimize the download time for those wishing to work with only one part of the tree. The two files are `sys.tar.gz`, which contains the files used to create the kernel, and `src.tar.gz` which contains all the other "user-land" utilities except the ports tree and the X11 sources. In general, however, you will usually want both of them installed. Assuming the downloaded files, `src.tar.gz` and `sys.tar.gz`, are in `/usr`:

```
# cd /usr/src
# tar xzf ../sys.tar.gz
# tar xzf ../src.tar.gz
# cd /usr
# tar xzf xenocara.tar.gz
# tar xzf ports.tar.gz
```

Not all people will wish to unpack all the file sets, but as the system must be kept in sync, you will generally need to set up all parts of the tree.

Common CVS tips

As indicated earlier, some options are mandatory to get a valid src tree in OpenBSD. The "`-P`" option above is one of those: It "prunes" (removes) directories that are empty. Over the years, directories have been created and deleted in the OpenBSD source tree, and sometimes the names of old directories are currently used as file names. Without the "`-P`" option, your newly checked-out tree WILL NOT successfully compile.

Much the same with the `-d` option on the 'update' command – it creates new directories that may have been added to the tree since your initial checkout. To get a successful update, you must use the `-Pd` options.

Experienced CVS users may wonder why the `CVSROOT` was specified and used in this example, as `cvs(1)` will record the CVS server's location in the checked out tree. This is correct, however there are enough times where one may need to override the default anoncvs server, many people recommend always specifying

the repository explicitly. It is also worth noting that while the CVSROOT environment variable can be used directly by cvs(1), it is used only if nothing else overrides it (i.e., cvs(1) would have an error without it), whereas specifying it in the cvs(1) command line overrides all other values.

It is often useful to use a .cvsrc in your home directory to specify defaults for some of these options. An example .cvsrc file:

```
$ more ~/.cvsrc
cvs -q -danoncvs@anoncvs.example.org:/cvs
diff -up
update -Pd
checkout -P
```

This file would cause cvs(1) to use the anoncvs@anoncvs.example.org:/cvs server, suppress usually unneeded output ("-q" is "quiet") for all operations, a "cvs up" command defaults to using the -Pd, a "cvs diff" defaults to providing "unified diffs" due to the "-u", and a "cvs checkout" will use the "-P" option. While this is convenient, if you forget this file exists, or try to run commands you got used to on a machine without this file, you will have problems.

As the source trees consist of large numbers of mostly small files, turning on soft updates for the partition the source tree is on will often give significantly better performance.

5.3.4 Building the kernel

We will assume you wish to build a standard (GENERIC or GENERIC.MP) kernel here. Normally, this is what you want to do. Do not consider building a custom kernel if you have not mastered the standard building process.

Obviously, the kernel is a VERY hardware dependent portion of the system. The source for the kernel is in the /usr/src/sys directory. Some parts of the OpenBSD kernel code are used on all platforms, others are very specific to one processor or one architecture. If you look in the /usr/src/sys/arch/ directory, you may see some things that look a little confusing – for example, there are mac68k, m68k and mvme68k directories. In this case, the mvme68k and mac68k systems both use the same processor, but the machines they are based on are very different, and thus require a very different kernel (there is much more to a computer's design than its processor!). However, parts of the kernel are common, those parts are kept in the m68k directory. If you are simply building a kernel, the base architecture directories like m68k are not anything for you to worry about, you will be working exclusively with the "compound architecture" directories, such as mvme68k.

Kernels are built based on kernel configuration files, which are located in the /usr/src/sys/arch/<your platform>/conf directory. Building the kernel consists of using the config(8) program to create and populate a kernel compile directory, which will end up in /usr/src/sys/arch/<your platform>/compile/<KernelName>. For this example, we will assume you are using the i386 platform:

```
# cd /usr/src/sys/arch/i386/conf
# config GENERIC
# cd ../compile/GENERIC
# make clean && make
    [...lots of output...]
# make install
```

Replace "`i386`" in the first line with your platform name. The `machine(1)` command can tell you what your platform name is, so an obvious generalization would be to use the command "`cd /usr/src/sys/arch/‘machine‘/conf`" instead on the first line.

At this point, reboot your machine to activate this new kernel. Note that the new kernel should be running before the next step, though if you have followed the above advice about upgrading to the most recent available snapshot, it may not matter as much. Sometimes, however, APIs change, and the old kernel will be unable to run new applications, but the new kernel will generally support the old ones.

Note that you can build a kernel without root access, but you must have root to install the kernel.

5.3.5 Building the userland

There is a specific process used by OpenBSD. Processes used on other OSs you may have been familiar with will most likely not work on OpenBSD, and will get you laughed at when you ask why.

- Clear your /usr/obj directory and rebuild symbolic links:

  ```
  # rm -rf /usr/obj/*
  # cd /usr/src
  # make obj
  ```

  Note that the use of the `/usr/obj` directory is mandatory. Failing to do this step before building the rest of the tree will likely leave your `src` tree in bad shape.

- Make sure all the appropriate directories are created.

  ```
  # cd /usr/src/etc && env DESTDIR=/ make distrib-dirs
  ```

- Build the system:

  ```
  # cd /usr/src
  # make build
  ```

This compiles and installs all the "userland" utilities in the appropriate order. This is a fairly time consuming step – a very fast machine may be able to complete it in well under an hour, a very slow machine may take many days. When this step is complete, you have newly compiled binaries in place on your system.

- **If building *-current***: Update `/dev` and `/etc`, with the changes listed in `current.html`. If following *-stable* after a proper upgrade process or a install of the proper starting binary, this step is not needed or desired.

5.4 Building a Release

What is a "release", and why would I want to make one?

A release is the complete set of files that can be used to install OpenBSD on another computer. If you have only one computer running OpenBSD, you really don't have any reason to make a release, as the above build process will do everything you need. An example use of the release process would be to build *-stable* on a fast machine, then make a release to be installed on all your other machines in your office.

The release process uses the binaries created in the */usr/obj* directory in the building process above, so you must successfully complete the build first, and nothing must disturb the */usr/obj* directory. A time where this might be a problem is if you use a memory disk as your */usr/obj* for a little extra performance in the build process, you would not want to reboot the computer between the "*build*" and "*release*" steps!

The release process requires two work directories, which we will call `DESTDIR` and `RELEASEDIR`. All the files that are part of a "`clean`" OpenBSD install will be copied to their proper place within the `DESTDIR`. They will then be `tar(1)`ed up and placed in the `RELEASEDIR`. At the end of the process, `RELEASEDIR` will hold the completed OpenBSD release. The release process will also use the */mnt* location, so this should not be used by anything while the release process is running. For the purpose of example, we will use the `DESTDIR` of */usr/dest* and the `RELEASEDIR` of */usr/rel*.

The release process involves a utility, *crunchgen(8)*, which is used to create a single executable file made up of many individual binaries. The name this single executable file is invoked by determines which component binary is run. This is how a number of individual program files are squeezed into the ramdisk kernel that exists on boot floppies and other boot media.

You must have root privileges to make a release.

Doing a release

Define our `DESTDIR` and `RELEASEDIR` environment variables:

```
# export DESTDIR=/usr/dest
```

```
# export RELEASEDIR=/usr/rel
```

We now clear the DESTDIR and create the directories if needed:

```
# test -d $DESTDIR && mv $DESTDIR $DESTDIR.old && rm -rf $DESTDIR.old &
# mkdir -p $DESTDIR $RELEASEDIR
```

RELEASEDIR does not normally need to be empty before starting the release process, however, if there are changes in the release files or their names, old files may be left laying around. You may wish to also erase this directory before starting.

We now make the release itself:

```
# cd /usr/src/etc
# make release
```

After the release is made, it is a good idea to check the release to make sure the tar files are matching what is in the DESTDIR. The output of this step should be very minimal.

```
# cd /usr/src/distrib/sets
# sh checkflist
```

You now have complete and checked release file sets in the RELEASEDIR. These files can now be used to install or upgrade OpenBSD on other machines.

The authoritative instructions on making a release are in release(8).

Note: if you wish to distribute the resultant files by HTTP for use by the upgrade or install scripts, you will need to add an "index.txt" file, which contains the list of all the files in your newly created release.

```
# /bin/ls -1 >index.txt
```

Once you have the complete release made, you can use those files for a standard install or upgrade on another machine, or if updating a machine to a new *-stable*, simply unpack the tar files in the root directory of the target machine.

5.5 Building X (Xenocara)

Starting with X.org v7, X switched to "modular build" system, splitting the x.org source tree into more than three hundred more-or-less independent packages.

To simplify life for OpenBSD users, a "meta-build" called Xenocara was developed. This system "converts" X back into one big tree to be built in one process. As an added bonus, this build process is much more similar to the build process used by the rest of OpenBSD than the previous versions were.

The official instructions for building X exist in your machine's /usr/xenocara/README file and in release(8).

Getting source code

The "usual" location for the xenocara source tree is `/usr/xenocara`, and the source is stored in the `xenocara` module in CVS. So, the checkout process is this:

```
$ cd /usr
$ cvs -qdanoncvs@anoncvs.example.org:/cvs checkout -P xenocara
```

Building Xenocara

For building the standard xenocara tree as supported by OpenBSD, no external tools are needed.

```
# cd /usr/xenocara
# rm -rf /usr/xobj/*
# make bootstrap
# make obj
# make build
```

If you wish to make actual modifications to the source code, you will probably need to add several packages. Details are in the `/usr/xenocara/README` file.

Making a release

This is similar to the main system release process. After successfully building X, you will define a `DESTDIR` and `RELEASEDIR`, with the same purposes as above. The `RELEASEDIR` can be the same directory as the main system `RELEASEDIR`, but `DESTDIR` will be erased and rebuilt in this process. If done carefully, this is not a problem, but using a separate `DESTDIR` may be "safer".

For this example, we will use a `DESTDIR` and `RELEASEDIR` of `/usr/dest` and `/usr/rel`, respectively. This must be done after the above build process.

```
# export DESTDIR=/usr/dest
# export RELEASEDIR=/usr/rel
# test -d $DESTDIR && mv $DESTDIR $DESTDIR- &&
     rm -rf $DESTDIR- &
# mkdir -p $DESTDIR $RELEASEDIR
# make release
```

When this process is completed, you will have a set of release files in the `$RELEASEDIR`.

5.6 Why do I need a custom kernel?

Actually, you probably don't.

A custom kernel is a kernel built with a configuration file other than the provided `GENERIC` configuration file. A custom kernel can be based on *-release*, *-stable* or *-current* code, just as a *GENERIC* kernel can be. While compiling your own *GENERIC* kernel is supported by the OpenBSD team, compiling your own custom kernel is *not*.

The standard OpenBSD kernel configuration (`GENERIC`) is designed to be suitable for most people. More people have broken their system by trying to tweak their kernel than have improved system operation. There are some people that believe that you must customize your kernel and system for optimum performance, but this is not true for OpenBSD. Only the most advanced and knowledgeable users with the most demanding applications need to worry about a customized kernel or system.

Some reasons you might want or need to build a custom kernel:

- You really know what you are doing, and want to shoe-horn OpenBSD onto a computer with a small amount of RAM by removing device drivers you don't need.
- You really know what you are doing, and wish to remove default options or add options which may not have been enabled by default (and have good reason to do so).
- You really know what you are doing, and wish to enable experimental options.
- You really know what you are doing, and have a special need that is not met by GENERIC, and aren't going to ask why it doesn't work if something goes wrong.

Some reasons why you should not build a custom kernel:

- You do not need to, normally.
- You will not get a faster system.
- You are likely to make a less reliable machine.
- You will not get any support from developers.
- You will be expected to reproduce any problem with a GENERIC kernel before developers take any problem report seriously.
- Users and developers will laugh at you when you break your system.
- Custom compiler options usually do a better job of exposing compiler problems than improving system performance.

Removing device drivers may speed the boot process on your system, but can complicate recovery should you have a hardware problem, and is very often done wrong. Removing device drivers *will not* make your system run faster

by any noticeable amount, though can produce a smaller kernel. Removing debugging and error checking can result in a measurable performance gain, but will make it impossible to troubleshoot a system if something goes wrong.

Again, developers will usually ignore bug reports dealing with custom kernels, unless the problem can be reproduced in a GENERIC kernel as well. You have been warned.

5.7 Building a custom kernel

It is assumed you have read the above, and really enjoy pain. It is also assumed that you have a goal that can not be achieved by either a Boot time configuration (UKC>) or by config(8)ing a GENERIC kernel. If all of this is not true, you should stick to using GENERIC. Really.

OpenBSD kernel generation is controlled by configuration files, which are located in the /usr/src/sys/arch/<arch>/conf/ directory by default. All architectures have a file, GENERIC, which is used to generate the standard OpenBSD kernel for that platform. There may also be other configuration files which are used to create kernels with different focuses, for example, for minimal RAM, diskless workstations, etc.

The configuration file is processed by config(8), which creates and populates a compilation directory in ../compile, on a typical installation, that would be in /usr/src/sys/arch/¡arch¿/compile/. config(8) also creates a Makefile, and other files required to successfully build the kernel.

Kernel Configuration Options are options that you add to your kernel configuration that place certain features into your kernel. This allows you to have exactly the support you want, without having support for unneeded devices. There are a multitude of options that allow you to customize your kernel. Here we will go over only some of them, those that are most commonly used. Check the options(4) man page for a complete list of options, and as these change from time to time, you should make sure you use a man page for the same version of OpenBSD you are building. You can also check the example configuration files that are available for your architecture.

Do not add, remove, or change options in your kernel unless you actually have a reason to do so! Do not edit the GENERIC configuration file!! The only kernel configuration which is supported by the OpenBSD team is the GENERIC kernel, the combination of the options in /usr/src/sys/arch/<arch>/conf/GENERIC and /usr/src/sys/conf/GENERIC *as shipped by the OpenBSD team* (i.e., NOT edited). Reporting a problem on a customized kernel will almost always result in you being told to try to reproduce the problem with a GENERIC kernel. Not all options are compatible with each other, and many options are required for the system to work. There is no guarantee that just because you manage to get a custom kernel compiled that it will actually run. There is no guarantee that a kernel that can be "config(8)ed" can be built.

You can see the platform-specific configuration files here:

- alpha Kernel Configuration Files

- i386 Kernel Configuration Files
- macppc Kernel Configuration Files
- sparc Kernel Configuration Files
- sparc64 Kernel Configuration Files
- vax Kernel Configuration Files
- hppa Kernel Configuration Files
- Other Arch's

Look closely at these files and you will notice a line near the top similar to:

```
include "../../../conf/GENERIC"
```

This means that it is referencing another configuration file, one that stores platform-independent options. When creating your kernel configuration, be sure to look through `sys/conf/GENERIC`.

Kernel configuration options should be placed in your kernel configuration file in the format of:

```
option      name
```

or

```
option      name=value
```

For example, to place option "`DEBUG`" in the kernel, add a line like this:

```
option      DEBUG
```

Options in the OpenBSD kernel are translated into compiler preprocessor options, therefore an option like `DEBUG` would have the source compiled with option `-DDEBUG`, which is equivalent to doing a `#define DEBUG` throughout the kernel.

Sometimes, you may wish to disable an option that is already defined, typically in the "`src/sys/conf/GENERIC`" file. While you could modify a copy of that file, a better choice would be to use the rmoption statement. For example, if you really wanted to disable the in-kernel debugger (*not recommended!*), you would add a line such as:

```
rmoption DDB
```

in your kernel configuration file. `option DDB` is defined in `src/sys/conf/GENERIC`, but the above `rmoption` line deactivates it.

Once again, please see `options(4)` for more information about the specifics of these options. Also note that many of the options also have their own manual pages – always read everything available about an option before adding or removing it from your kernel.

Building a custom kernel

In this case, we will build a kernel that supports the boca(4) ISA multi-port serial card. This card is not in the GENERIC kernel, due to conflicts with other drivers. There are two common ways to make a custom kernel: copy the GENERIC config file to another name and edit it, or create a "wrapper" file that "includes" the standard GENERIC kernel and any options you need that aren't in GENERIC. In this case, our wrapper file looks like this:

```
include "arch/i386/conf/GENERIC"

boca0  at       isa? port 0x100 irq 10     # BOCA 8-port serial cards
com*   at       boca? slave ?
```

The two lines regarding the boca(4) card are copied from the commented out lines in GENERIC, with the IRQ adjusted as needed. The advantage to using this "wrapper" file is any unrelated changes in GENERIC are updated automatically with any other source code update. The disadvantage is one can not remove devices (though in general, that's a bad idea, anyway).

The other way to generate a custom kernel is to make a copy of the standard GENERIC, giving it another name, then editing it as needed. The disadvantage to this is later updates to the GENERIC configuration file have to be merged into your copy, or you have to remake your configuration file.

In either event, after making your custom kernel configuration file, use config(8) and make the kernel as documented above.

Full instructions for creating your own custom kernel are in the config(8) man page.

5.8 Boot-Time Configuration

Sometimes when booting your system you might notice that the kernel finds your device but maybe at the wrong IRQ. And maybe you need to use this device right away. Well, without rebuilding the kernel you can use OpenBSD's boot time kernel configuration. This will only correct your problem for one time. If you reboot, you will have to repeat this procedure. So, this is only meant as a temporary fix, and you should correct the problem using config(8). Your kernel does however need **option BOOT_CONFIG** in the kernel, which GENERIC does have.

Most of this document can be found in the man page boot_config(8).

To boot into the User Kernel Config, or UKC, use the -c option at boot time.

```
boot> boot hd0a:/bsd -c
```

Or whichever kernel it is you want to boot. Doing this will bring up a UKC prompt. From here you can issue commands directly to the kernel specifying devices you want to change or disable or even enable.

Here is a list of common commands in the UKC.

- `add device` - Add a device through copying another
- `change devno | device` - Modify one or more devices
- `disable devno | device` - Disable one or more devices
- `enable devno | device` - Enable one or more devices
- `find devno | device` - Find one or more devices
- `help` - Short summary of these commands
- `list` - List ALL known devices
- `exit/quit` - Continue Booting
- `show [attr [val]]` - Show devices with an attribute and optional with a specified value

Once you have your kernel configured, use `quit` or `exit` and continue booting. After doing so, you should make the change permanent in your kernel image, as described in Using `config(8)` to change your kernel.

5.9 Using `config(8)` to change your kernel

The `-e` and `-u` options with `config(8)` can be extremely helpful and save wasted time compiling your kernel. The `-e` flag allows you to enter the `UKC` or User Kernel Config on a running system. These changes will then take place on your next reboot. The `-u` flag tests to see if any changes were made to the running kernel during boot, meaning you used `boot -c` to enter the `UKC` while booting your system.

The following example shows the disabling of the `ep*` devices in the kernel. For safety's sake you must use the `-o` option which writes the changes out to the file specified. For example : `config -e -o bsd.new /bsd` will write the changes to `bsd.new`. The example doesn't use the `-o` option, therefore changes are just ignored, and not written back to the kernel binary. For more information pertaining to error and warning messages read the `config(8)` man page.

```
$ sudo config -e /bsd
OpenBSD 5.0 (GENERIC) #43: Wed Aug 17 10:10:52 MDT 2011
    deraadt@i386.openbsd.org:/usr/src/sys/arch/i386/compile/GENERIC
warning: no output file specified
Enter 'help' for information
ukc> ?
        help                                    Command help list
        add             dev                     Add a device
        base            8|10|16                 Base on large numbers
        change          devno|dev               Change device
```

```
        disable         attr val|devno|dev  Disable device
        enable          attr val|devno|dev  Enable device
        find            devno|dev           Find device
        list                                List configuration
        lines           count               # of lines per page
        show            [attr [val]]        Show attribute
        exit                                Exit, without saving changes
        quit                                Quit, saving current changes
        timezone        [mins [dst]]        Show/change timezone
        bufcachepercent [number]            Show/change BUFCACHEPERCENT
        nkmempg         [number]            Show/change NKMEMPAGES
ukc> list
  0 video* at uvideo* flags 0x0
  1 audio* at uaudio*|sb0|sb*|gus0|pas0|ess*|wss0|wss*|ym*|eap*|envy*|eso*|sv*|n
eo*|cmpci*|clcs*|clct*|auacer*|auglx*|auich*|auixp*|autri*|auvia*|azalia*|fms*|m
aestro*|esa*|yds*|emu* flags 0x0
  2 midi* at umidi*|sb0|sb*|ym*|mpu*|mpu*|autri*|eap*|envy* flags 0x0
  3 drm* at inteldrm*|radeondrm* flags 0x0
  4 inteldrm* at vga0|vga* flags 0x0
  5 radeondrm* at vga0|vga* flags 0x0
  6 radio* at udsbr*|bktr0|fms* flags 0x0
  7 vscsi0 at root flags 0x0
  8 softraid0 at root flags 0x0
  9 nsphy* at url*|udav*|mos*|axe*|aue*|xe*|ef*|hme*|lii*|bce*|ale*|alc*|age*|jm
e*|et*|nfe*|stge*|vge*|bnx*|bge*|lge*|nge*|msk*|sk*|ste*|se*|sis*|wb*|tl*|vte*|v
r*|pcn*|sf*|ti*|gem*|ne0|ne1|ne2|ne*|ne*|ne*|epic*|sm0|sm*|dc*|dc*|re*|re*|rl*|r
l*|mtd*|fxp*|fxp*|xl*|xl*|ep0|ep*|ep*|ep*|ep*|ep* phy -1 flags 0x0
 10 nsphyter* at url*|udav*|mos*|axe*|aue*|xe*|ef*|hme*|lii*|bce*|ale*|alc*|age*
|jme*|et*|nfe*|stge*|vge*|bnx*|bge*|lge*|nge*|msk*|sk*|ste*|se*|sis*|wb*|tl*|vte
*|vr*|pcn*|sf*|ti*|gem*|ne0|ne1|ne2|ne*|ne*|ne*|epic*|sm0|sm*|dc*|dc*|re*|re*|rl
*|rl*|mtd*|fxp*|fxp*|xl*|xl*|ep0|ep*|ep*|ep*|ep*|ep* phy -1 flags 0x0
[...snip...]
ukc> disable ep
95 ep* disabled
 96 ep* disabled
281 ep0 disabled
282 ep* disabled
283 ep* disabled
340 ep* disabled
ukc> quit
not forced
```

In the above example, all ep* devices are disabled in the kernel and will not be probed. In some situations where you have used the UKC during boot, via boot -c, you will need these changes to be written out permanently. To do this you need to use the -u option. In the following example, the computer

was booted into the UKC and the wi(4) device was disabled. Since changes made with boot -c are NOT permanent, these changes must be written out. This example writes the changes made from boot -c into a new kernel binary bsd.new.

```
$ sudo config -e -u -o bsd.new /bsd
OpenBSD 5.0 (GENERIC) #43: Wed Aug 17 10:10:52 MDT 2011
    deraadt@i386.openbsd.org:/usr/src/sys/arch/i386/compile/GENERIC
Processing history...
161 wi* disabled
162 wi* disabled
417 wi* disabled
Enter 'help' for information
ukc> quit
```

5.10 Getting more verbose output during boot

Getting more verbose output can be very helpful when trying to debug problems when booting. If you have a problem wherein your boot floppy won't boot and need to get more information, simply reboot. When you get to the "boot>" prompt, boot with boot -c. This will bring you into the UKC>, then do:

```
UKC> verbose
autoconf verbose enabled
UKC> quit
```

Now you will be given extremely verbose output upon boot.

5.11 Common problems, tips and questions when compiling and building

Most of the time, problems in the build process are caused by not following the above directions carefully. There are occasional real problems with building *-current* from the most recent snapshot, but failures when building *-release* or *-stable* are almost always user error.

Most problems are usually one of the following:

- Failing to start from the appropriate binary, including attempting to upgrade from source or assuming a week old snapshot is "close enough".
- Checking out the wrong branch of the tree.
- Not following the process.
- Trying to customize or "optimize" your system.

Here are some additional problems you might encounter, however:

5.11.1 The build stopped with a "Signal 11" error

Building OpenBSD and other programs from source is a task which pushes hardware harder than most others, making intensive use of CPU, disk and memory. As a result, if you have hardware which has a problem, the most likely time for that problem to appear is during a build. `Signal 11` failures are *typically* caused by hardware problems, very often memory problems, but can also be CPU, main board, or heat issues. Your system may actually be very stable otherwise, but unable to compile programs.

You will probably find it best to repair or replace the components that are causing trouble, as problems may show themselves in other ways in the future. If you have hardware which you really wish to use and causes you no other problem, simply install a snapshot or a release.

For much more information, see the `Sig11` FAQ.

5.11.2 "make build" fails with "cannot open output file snake: is a directory"

This is the result of two separate errors:

- **You did not fetch or update your CVS tree properly**. When doing a CVS checkout operation, you must use the "-P" option, when you update your source tree with CVS, you must use "-Pd" options to cvs(1), as documented above. These options make sure new directories are added and removed from the tree as OpenBSD evolves.
- **You did not properly create the obj directory before your build**. Building the tree without a /usr/obj directory is not supported.

It is important to carefully follow the instructions when fetching and building your tree.

5.11.3 My IPv6-less system doesn't work!

Yes. Please do not make modifications to the base system that you don't understand the implications of. One "little" change in the kernel can have very large impact to the entire rest of the system. Please re-read this.

5.11.4 Oops! I forgot to make the /usr/obj directory first!

By doing a "`make build`" before doing a "`make obj`", you will end up with the object files scattered in your `/usr/src` directory. This is a bad thing. If you wish to try to avoid re-fetching your entire src tree again, you can try the following to clean out obj files:

```
# cd /usr/src
# find . -type l -name obj | xargs rm
```

```
# make cleandir
# rm -rf /usr/obj/*
# make obj
```

5.11.5 Tip: Put /usr/obj on its own partition

If you build often, you may find it faster to put `/usr/obj` on its own partition. The benefit is simple, it is typically faster to:

```
# umount /usr/obj
# newfs YourObjPartition
# mount /usr/obj
```

than to "`rm -rf /usr/obj/*`".

5.11.6 How do I not build parts of the tree?

Sometimes, you may wish to not build certain parts of the tree, typically because you have installed a replacement for an included application from packages, or wish to make a "smaller" release for whatever reason. The solution to this is to use the `SKIPDIR` option of `/etc/mk.conf`.

Note: it is possible to make a broken system this way. The results of this option are not supported by the OpenBSD project.

5.11.7 Where can I learn more about the build process?

Here are some other resources:

- `release(8)`
- `config(8)`
- `mk.conf(5)`
- `options(4)`
- `/usr/src/Makefile`
- `Patch Branches` (*-stable*)
- (for X) `/usr/X11R6/README` on your installed system

5.11.8 I didn't see any snapshots on the FTP site. Where did they go?

Snapshots may be removed as they become old (or no longer relevant) or near the time of a new *-release*.

5.11.9 How do I bootstrap a newer version of the compiler (*gcc*)?

You should really just install the latest snapshot.

5.11.10 What is the best way to update /etc, /var, and /dev?

As a policy, software in the OpenBSD tree does not modify files in `/etc` automatically. This means it is *always* up to the administrator to make the necessary modifications there. Upgrades are no exception. To update files in these directories, first determine what changes have occurred to the base (distribution) files, and then manually reapply these changes.

For example, to see the files in the tree that have changed most recently, do a:

```
# cd /usr/src/etc
# ls -lt |more
```

To see all the changes in `/etc` between arbitrary versions of OpenBSD, you can use CVS. For example, to see the changes between 4.9 and 5.0 do a:

```
# cd /usr/src/etc
# cvs diff -u -rOPENBSD_4_9 -rOPENBSD_5_0
```

To see the changes between 5.0 and *-current* ("HEAD"), use:

```
# cd /usr/src/etc
# cvs diff -u -rOPENBSD_5_0-rHEAD
```

The `/dev/MAKEDEV` script is not updated automatically as part of the make build process, however it is installed as part of a binary upgrade. As a general rule, it is a good idea to copy (if needed) and run this script from your source tree when performing an upgrade:

```
# cd /dev
# cp /usr/src/etc/etc.‘machine‘/MAKEDEV ./
# ./MAKEDEV all
```

Once you have identified the changes, reapply them to your local tree, preserving any local configuration you may have done.

Typical `/etc` changes to watch out for between releases include:

- Additions to `/etc/protocols` and `/etc/services`
- New sysctls (see `/etc/sysctl.conf`)
- Changes to the default cron jobs. See `/etc/daily`, `/etc/weekly`, `/etc/monthly`, and `/etc/security`

- All rc scripts, including netstart
- Device changes, see above
- File hierarchy changes in `/etc/mtree`, see below
- New users (`/etc/passwd`) and groups (`/etc/group`)

These changes are summarized in `upgrade50.html` (for going to 5.0-release) or `current.html` (for going to *-current*).

5.11.11 Is there an easy way to make all the file hierarchy changes?

From time to time, files or directories are added to, or removed from the file hierarchy. Also, ownership information for portions of the filesystem may change. An easy way to ensure that your file hierarchy is up-to-date is to use the `mtree(8)` utility.

First, fetch the latest source, then do the following:

```
# cd /usr/src/etc/mtree
# install -c -o root -g wheel -m 600 special /etc/mtree
# install -c -o root -g wheel -m 444 4.4BSD.dist /etc/mtree
# mtree -qdef /etc/mtree/4.4BSD.dist -p / -u
```

Your file hierarchy should now be up to date.

5.11.12 Can I cross-compile? Why not?

Cross-compiling tools are in the system, for use by developers bringing up a new platform. However, they are not maintained for general use.

When the developers bring up support for a new platform, one of the first big tests is a native-build. Building the system from source puts considerable load on the OS and machine, and does a very good job of testing how well the system really works. For this reason, OpenBSD does all the build process on the platform the build is being used for, also known as "native building". Without native building, it is much more difficult to be sure that the various platforms are actually running reliably, and not just booting.

Chapter 6

Networking

6.1 Before we go any further

For the bulk of this document, it helps if you have read and at least partially understood the Kernel Configuration and Setup section of the FAQ, and the `ifconfig(8)` and `netstat(1)` man pages.

If you are a network administrator, and you are setting up routing protocols, if you are using your OpenBSD box as a router, if you need to go in depth into IP networking, you really need to read Understanding IP Addressing. This is an excellent document. "Understanding IP Addressing" contains fundamental knowledge to build upon when working with IP networks, especially when you deal with or are responsible for more than one network.

If you are working with applications such as web servers, ftp servers, and mail servers, you may benefit greatly by reading the RFCs. Most likely, you can't read all of them. Pick some topics that you are interested in, or that you use in your network environment. Look them up, find out how they are intended to work. The RFCs define many (thousands of) standards for protocols on the Internet and how they are supposed to work.

6.2 Network configuration

Normally, OpenBSD is initially configured by the installation process. However, it is good to understand what is happening in this process and how it works. All network configuration is done using simple text files in the `/etc` directory.

6.2.1 Identifying and setting up your network interfaces

In OpenBSD, interfaces are named for the type of card, not for the type of connection. You can see your network card get initialized during the booting process, or after the booting process using the `dmesg(8)` command. You also have the chance of seeing your network interface using the `ifconfig(8)` command. For example, here is the output of dmesg for a Intel Fast Ethernet network card, which uses the device name `fxp`.

```
fxp0 at pci0 dev 10 function 0 "Intel 82557" rev 0x0c: irq 5, address 00:02:b3:2b:10:f
inphy0 at fxp0 phy 1: i82555 10/100 media interface, rev. 4
```

If you don't know what your device name is, please look at the supported hardware list for your platform. You will find a list of many common card names and their OpenBSD device names here. Combine the short alphabetical device name (such as `fxp`) with a number assigned by the kernel and you have an interface name (such as `fxp0`). The number is assigned based on various criteria, depending upon the card and other details of the system. Some cards are assigned by the order they are found during bus probing. Others may be by hardware resource settings or MAC address.

You can find out what network interfaces have been identified by using the `ifconfig(8)` utility. The following command will show all network interfaces

on a system. This sample output shows us only one physical Ethernet interface, an `fxp(4)`.

```
$ ifconfig
lo0: flags=8049<UP,LOOPBACK,RUNNING,MULTICAST> mtu 33200
        priority: 0
        groups: lo
        inet6 ::1 prefixlen 128
        inet6 fe80::1%lo0 prefixlen 64 scopeid 0x3
        inet 127.0.0.1 netmask 0xff000000
fxp0: flags=8843<UP,BROADCAST,RUNNING,SIMPLEX,MULTICAST> mtu 1500
        lladdr 00:04:ac:dd:39:6a
        priority: 0
        media: Ethernet autoselect (100baseTX full-duplex)
        status: active
inet 192.168.1.34 netmask 0xffffff00 broadcast 192.168.1.255
        inet6 fe80::204:acff:fedd:396a%fxp0 prefixlen 64 scopeid 0x1
enc0: flags=0<>
        priority: 0
        groups: enc
        status: active
pflog0: flags=141<UP,RUNNING,PROMISC> mtu 33200
        priority: 0
        groups: pflog
```

As you can see here, `ifconfig(8)` gives us a lot more information than we need at this point. But, it still allows us to see our interface. In the above example, the interface card is already configured. This is obvious because an IP network is already configured on `fxp0`, hence the values "`inet 10.0.0.38 netmask 0xffffff00 broadcast 10.0.0.255`". Also, the `UP` and `RUNNING` flags are set.

Finally, you will notice several other interfaces come enabled by default. These are virtual interfaces that serve various functions. The following manual pages describe them:

- `lo` - Loopback Interface
- `pflog` - Packet Filter Logging Interface
- `enc` - Encapsulating Interface

Other virtual interfaces are automatically created on-demand, including:

- `sl` - SLIP Network Interface
- `ppp` - Point to Point Protocol
- `tun` - Tunnel Network Interface
- `bridge` - Ethernet Bridge Interface

- `vlan` - IEEE 802.1Q Encapsulation Interface
- `svlan` - IEEE 802.1AD Provider Bridges (QinQ)
- `gre` - GRE/MobileIP Encapsulation Interface
- `gif` - Generic IPv4/IPv6 Tunnel Interface
- `carp` - Common Address Redundancy Protocol Interface
- `mpe` - MPLS Provider Edge
- `vether` - Virtual Ethernet Interface

Interfaces are configured at boot time using /etc/hostname.*if* files, where *if* is replaced by the full name of each interface and each interface has its own file. The example above would use the file `/etc/hostname.fxp0`.

The layout of this file is simple:

```
address_family address netmask broadcast [other options]
```

Much more detail about the format of this file can be found in the `hostname.if(5)` man page. You will need to read this for less trivial configurations.

A typical interface configuration file, configured for an IPv4 address, would look like this:

```
$ cat /etc/hostname.fxp0
inet 10.0.0.38 255.255.255.0 NONE
```

In this case, we have defined an IPv4 (inet) address, with an IP address of 10.0.0.38, a subnet mask of 255.255.255.0 and no specific broadcast address (which will default to 10.0.0.255 in this case).

You could also specify media types for Ethernet, say, if you wanted to force 100baseTX full-duplex mode.

```
inet 10.0.0.38 255.255.255.0 NONE media 100baseTX mediaopt full-duplex
```

(Of course, you should never force full duplex mode unless both sides of the connection are set to do this! In the absence of special needs, media settings should be excluded. A more likely case might be to force 10base-T or half duplex when your infrastructure requires it.)

Or, you may want to use special flags specific to a certain interface. The format of the hostname file doesn't change much!

```
$ cat /etc/hostname.vlan0
inet 172.21.0.31 255.255.255.0 NONE vlan 2 vlandev fxp1
```

6.2.2 Default gateway

Put the IP of your gateway in the file */etc/mygate*. This will allow for your gateway to be set upon boot. This file consists of one line, with just the address of this machine's gateway address:

```
10.0.0.1
```

It is possible to use a symbolic name there, but be careful: you can't assume things like the resolver are fully configured or even reachable until AFTER the default gateway is configured. In other words, it had better be an IP address or something that is defined in the `/etc/hosts` file.

6.2.3 DNS Resolution

DNS resolution is controlled by the file `/etc/resolv.conf`. Here is an example of a `/etc/resolv.conf` file:

```
search example.com
nameserver 125.2.3.4
nameserver 125.2.3.5
lookup file bind
```

In this case, the default domain name will be `example.com`, there are two DNS resolvers, `125.2.3.4` and `125.2.3.5` specified, and the `/etc/hosts` file will be consulted before the DNS resolvers are.

As with virtually all Unix (and many non-Unix) systems, there is an `/etc/hosts` file which can be used to specify systems that are not in (or if used with the above "lookup" priority, not as desired in) the formal DNS system.

If you are using DHCP, you'll want to read 6.4 - DHCP taking note of `resolv.conf.tail(5)`.

6.2.4 Host name

Every Unix machine has a name. In OpenBSD, the name is specified as a "Fully Qualified Domain Name" (FQDN) in one line in the file `/etc/myname`. If this machine is named "puffy" and in the domain "example.com", the file would contain the one line:

```
puffy.example.com
```

6.2.5 Activating the changes

From here, you can either reboot or run the `/etc/netstart` script. You can do this by simply typing (as root):

```
# sh /etc/netstart
writing to routing socket: File exists
add net 127: gateway 127.0.0.1: File exists
writing to routing socket: File exists
add net 224.0.0.0: gateway 127.0.0.1: File exists
```

Notice that a few errors were produced. By running this script, you are reconfiguring things which are already configured. As such, some routes already exist in the kernel routing table. From here your system should be up and running. Again, you can check to make sure that your interface was setup correctly with `ifconfig(8)`.

Even though you can completely reconfigure networking on an OpenBSD system without rebooting, a reboot is HIGHLY recommended after any significant reconfiguration. The reason for this is the environment at boot is somewhat different than it is when the system is completely up and running. For example, if you had specified a DNS-resolved symbolic name in any of the files, you would probably find it worked as expected after reconfigure, but on initial boot, your external resolver may not be available, so the configuration will fail.

6.2.6 Checking routes

You can check your routes via `netstat(1)` or `route(8)`. If you are having routing problems, you may want to use the `-n` flag to `route(8)` which prints the IP addresses rather than doing a DNS lookup and displaying the hostname. Here is an example of viewing your routing tables using both programs.

```
$ netstat -rn
Routing tables

Internet:
Destination        Gateway            Flags     Refs     Use    Mtu  Interface
default            10.0.0.1           UGS          0      86      -  fxp0
127/8              127.0.0.1          UGRS         0       0      -  lo0
127.0.0.1          127.0.0.1          UH           0       0      -  lo0
10.0.0/24          link#1             UC           0       0      -  fxp0
10.0.0.1           aa:0:4:0:81:d      UHL          1       0      -  fxp0
10.0.0.38          127.0.0.1          UGHS         0       0      -  lo0
224/4              127.0.0.1          URS          0       0      -  lo0

Encap:
Source             Port  Destination        Port  Proto SA(Address/SPI/Proto)

$ route show
Routing tables

Internet:
```

```
Destination      Gateway            Flags
default          10.0.0.1           UG
127.0.0.0        LOCALHOST          UG
localhost        LOCALHOST          UH
10.0.0.0         link#1             U
10.0.0.1         aa:0:4:0:81:d      UH
10.0.0.38        LOCALHOST          UGH
BASE-ADDRESS.MCA LOCALHOST          U
```

6.2.7 Setting up your OpenBSD box as a forwarding gateway

This is the basic information you need to set up your OpenBSD box as a gateway (also called a router). If you are using OpenBSD as a router on the Internet, we suggest that you also read the Packet Filter setup instructions below to block potentially malicious traffic. Also, due to the low availability of IPv4 addresses from network service providers and regional registries, you may want to look at Network Address Translation for information on conserving your IP address space.

The `GENERIC` kernel already has the ability to allow IP Forwarding, but needs to be turned on. You should do this using the `sysctl(8)` utility. To change this permanently you should edit the file `/etc/sysctl.conf` to allow for IP Forwarding. To do so add this line in that configuration file.

```
net.inet.ip.forwarding=1
```

To make this change without rebooting you would use the `sysctl(8)` utility directly. Remember though that this change will no longer exist after a reboot, and needs to be run as root.

```
# sysctl net.inet.ip.forwarding=1
net.inet.ip.forwarding: 0 -> 1
```

Now modify the routes on the other hosts on both sides. This is often done with static route entries, but more advanced networks can make use of the rich suite of routing daemons included as part of OpenBSD to pass routes to other systems. These include OpenBGPD, `ospfd(8)`, `ospf6d(8)`, `ldpd(8)` and `ripd(8)`. Additionally the ports collection includes software such as bird, igmpproxy and quagga. OpenBSD supports several T1, HSSI, ATM, FDDI and serial (PPP/SLIP) interfaces, and of course many Ethernet devices (including 10Gb).

6.2.8 Setting up aliases on an interface

OpenBSD has a simple mechanism for setting up IP aliases on an interface. To do this simply edit the file `/etc/hostname.⟨if⟩`. This file is read upon boot by the `/etc/netstart(8)` script, which is part of the `rc` startup hierarchy. For the

example, we assume that the user has an interface `dc0` and is on the network `192.168.0.0`. Other important information:

- IP for `dc0` is 192.168.0.2
- NETMASK is 255.255.255.0

A few side notes about aliases. In OpenBSD you use the interface name only. There is no difference between the first alias and the second alias. Unlike some other operating systems, OpenBSD doesn't refer to them as `dc0:0`, `dc0:1`. If you are referring to a specific aliased IP address with ifconfig, or adding an alias, be sure to say "`ifconfig int alias`" instead of just "`ifconfig int`" at the command line. You can delete aliases with "`ifconfig int delete`".

Assuming you are using multiple IP addresses which are in the same IP subnet with aliases, your netmask setting for each alias becomes 255.255.255.255. They do not need to follow the netmask of the first IP bound to the interface. In this example, `/etc/hostname.dc0`, two aliases are added to the device `dc0`, which, by the way, was configured as `192.168.0.2` netmask `255.255.255.0`.

```
# cat /etc/hostname.dc0
inet 192.168.0.2 255.255.255.0 NONE media 100baseTX
inet alias 192.168.0.3 255.255.255.255
inet alias 192.168.0.4 255.255.255.255
```

Once you've made this file, it just takes a reboot for it to take effect. You can, however, bring up the aliases by hand using the `ifconfig(8)` utility. To bring up the first alias you would use the command:

```
# ifconfig dc0 inet alias 192.168.0.3 netmask 255.255.255.255
```

(but again, a reboot is recommended to make sure you entered everything as you expected it to be!)

To view these aliases you must use the command:

```
$ ifconfig -A
dc0: flags=8863<UP,BROADCAST,NOTRAILERS,RUNNING,SIMPLEX,MULTICAST>
        media: Ethernet manual
        inet 192.168.0.2 netmask 0xffffff00 broadcast 192.168.0.255
        inet 192.168.0.3 netmask 0xffffffff broadcast 192.168.0.3
```

6.3 How do I filter and firewall with OpenBSD?

Packet Filter (from here on referred to as PF) is OpenBSD's system for filtering IP traffic and doing Network Address Translation. PF is also capable of normalizing and conditioning IP traffic and providing bandwidth control and packet prioritization, and can be used to create powerful and flexible firewalls. It is described in the PF User's Guide.

6.4 Dynamic Host Configuration Protocol (DHCP)

Dynamic Host Configuration Protocol is a way to configure network interfaces "automatically". OpenBSD can be a DHCP server (configuring other machines), a DHCP client (configured by another machine), and in some cases, can be both.

6.4.1 DHCP Client

To use the DHCP client `dhclient(8)` included with OpenBSD, edit `/etc/hostname.xl0` (this is assuming your main Ethernet interface is `xl0`. Yours might be `ep0` or `fxp0` or something else.) All you need to put in this hostname file is '`dhcp`':

```
# echo dhcp > /etc/hostname.xl0
```

This will cause OpenBSD to automatically start the DHCP client on boot. OpenBSD will gather its IP address, default gateway, and DNS servers from the DHCP server.

If you want to start a DHCP client from the command line, make sure `/etc/dhclient.conf` exists, then try:

```
# dhclient fxp0
```

Where `fxp0` is the interface on which you want to receive DHCP.

No matter how you start the DHCP client, you can edit the `/etc/dhclient.conf` file to not update your DNS according to the dhcp server's idea of DNS by first uncommenting the 'request' lines in it (they are examples of the default settings, but you need to uncomment them to override dhclient's defaults.)

```
request subnet-mask, broadcast-address, time-offset, routers,
      domain-name, domain-name-servers, host-name, lpr-servers, ntp-servers;
```

and then remove `domain-name-servers`. Of course, you may want to remove `host-name`, or other settings too.

By changing options in your `dhclient.conf(5)` file, you're telling the DHCP client how to build your `resolv.conf(5)` file. The DHCP client overrides any information you already have in `resolv.conf(5)` with the information it retrieves from the DHCP server. Therefore, you'll lose any changes you made manually to `resolv.conf`.

There are two mechanisms available to prevent this:

- `OPTION MODIFIERS` (default, supersede, prepend, and append) allow you to override any of the options in dhclient.conf(5).
- `resolv.conf.tail(5)` allows you to append anything you want to the resolv.conf(5) file created by dhclient(8).

An example would be if you're using DHCP but you want to append `lookup file bind` to the `resolv.conf(5)` created by `dhclient(8)`. There is no option for this in `dhclient.conf` so you must use `resolv.conf.tail` to preserve this.

```
# echo "lookup file bind" > /etc/resolv.conf.tail
```

Now your resolv.conf(5) should include "lookup file bind" at the end.

```
nameserver 192.168.1.1
nameserver 192.168.1.2
lookup file bind
```

6.4.2 DHCP Server

If you want to use OpenBSD as a DHCP server dhcpd(8), edit /etc/rc.conf.local so that it contains the line dhcpd_flags="*interface*", replacing interface with the list of interfaces that dhcpd(8) should listen on, for example:

```
# echo 'dhcpd_flags="xl1 xl2 xl3"' >>/etc/rc.conf.local
```

Then, edit /etc/dhcpd.conf. The options are pretty self-explanatory.

```
option  domain-name "example.com";
        option  domain-name-servers 192.168.1.3, 192.168.1.5;

        subnet 192.168.1.0 netmask 255.255.255.0
                option routers 192.168.1.1;

                range 192.168.1.32 192.168.1.127;
```

This will tell your DHCP clients that the domain to append to DNS requests is example.com (so, if the user types in 'telnet joe' then it will send them to joe.example.com). It will point them to DNS servers 192.168.1.3 and 192.168.1.5. For hosts that are on the same network as an Ethernet interface on the OpenBSD machine, which is in the 192.168.1.0/24 range, it will assign them an IP address between 192.168.1.32 and 192.168.1.127. It will set their default gateway as 192.168.1.1.

If you want to start dhcpd(8) from the command line, after editing /etc/dhcpd.conf, try:

```
# touch /var/db/dhcpd.leases
# dhcpd fxp0
```

The touch line is needed to create an empty dhcpd.leases file before dhcpd(8) can start. The OpenBSD startup scripts will create this file if needed on boot, but if you are starting dhcpd(8) manually, you must create it first. fxp0 is an interface that you want to start serving DHCP on.

If you are serving DHCP to a Windows box, you may want dhcpd(8) to give the client a 'WINS' server address. To make this happen, just add the following line to your /etc/dhcpd.conf:

```
option    netbios-name-servers    192.168.92.55;
```

(where 192.168.92.55 is the IP of your Windows or Samba server.) See dhcp-options(5) for more options that your DHCP clients may want.

6.5 PPP

The Point to Point Protocol (PPP) is generally what is used to create a connection to your ISP via a dial-up modem. OpenBSD has 2 ways of doing this:

- `pppd(8)` - the kernel PPP daemon
- `ppp(8)` - the userland PPP daemon

Both `ppp` and `pppd` perform similar functions, in different ways. `pppd` works with the kernel `ppp(4)` driver, whereas `ppp` works in userland with `tun(4)`. This document will cover only the userland PPP daemon, since it is easier to debug and to interact with. To start off you will need some simple information about your ISP. Here is a list of helpful information that you will need.

- Your ISP's dial-up number
- Your nameserver
- Your username and password
- Your gateway

Some of these you can do without, but would be helpful in setting up `ppp`. The userland PPP daemon uses the file `/etc/ppp/ppp.conf` as its configuration file. There are many helpful files in `/etc/ppp` that can have different setups for many different situations. You should take a browse through that directory.

Initial Setup - for `PPP(8)`

Initial Setup for the userland PPP daemon consists of editing your */etc/ppp/ppp.conf* file. This file doesn't exist by default, but there is a file */etc/ppp/ppp.conf.sample* which you can simply edit to create your own ppp.conf file. Here I will start with the simplest and probably most used setup. Here is a quick *ppp.conf* file that simply sets some defaults:

```
default:
  set log Phase Chat LCP IPCP CCP tun command
  set device /dev/cua01
  set speed 115200
  set dial "ABORT BUSY ABORT NO\\sCARRIER TIMEOUT 5 \"\" AT OK-AT-OK ATE1Q0 OK \\dATDT\\T TIMEOUT
```

The section under the `default`: tag gets executed each time. Here we set up all our critical information. With "set log" we set our logging levels. This can be changed: refer to `ppp(8)` for more info on setting up logging levels. Our device gets set with "set device". This is the device that the modem is on. In this example the modem is on com port 2. Therefore com port 1 would be `/dev/cua00`. With "`set speed`" we set the speed of our dial-up connection and

with "`set dial`" we set our dial-up parameters. With this we can change our timeout time, etc. This line should stay pretty much as it is though.

Now we can move on and set up information specific to our ISP. We do this by adding another tag under our `default`: section. This tag can be called anything you want - easiest to just use the name of your ISP. Here I will use `myisp`: as our tag referring to our ISP. Here is a simple setup incorporating all we need to get ourselves connected:

```
myisp:
  set phone 1234567
  set login "ABORT NO\\sCARRIER TIMEOUT 5 ogin:--ogin: ppp word: ppp"
  set timeout 120
  set ifaddr 10.0.0.1/0 10.0.0.2/0 255.255.255.0 0.0.0.0
  add default HISADDR
  enable dns
```

Here we have set up essential info for that specific ISP. The first option "`set phone`" sets your ISP's dial-up number. The "`set login`" sets our login options. Here we have the timeout set to 5; this means that we will abort our login attempt after 5 seconds if no carrier is found. Otherwise it will wait for "`login`:" to be sent and send in your username and password.

In this example our Username = ppp and Password = ppp. These values will need to be changed. The line "`set timeout`" sets the idle timeout for the entire connection duration to 120 seconds. The "`set ifaddr`" line is a little tricky. Here is a more extensive explanation.

```
set ifaddr 10.0.0.1/0 10.0.0.2/0 255.255.255.0 0.0.0.0
```

In the above line, we have it set in the format of "**`set ifaddr [myaddr[/nn] [hisaddr[/nn] [netmask [triggeraddr]]]]`**". So the first IP specified is what we want as our IP. If you have a static IP address, you set it here. In our example we use /0 which says that no bits of this IP address need to match and the whole thing can be replaced. The second IP specified is what we expect as their IP. If you know this you can specify it. Again in our line we don't know what will be assigned, so we let them tell us. The third option is our netmask, here set to `255.255.255.0`. If triggeraddr is specified, it is used in place of myaddr in the initial IPCP negotiation. However, only an address in the myaddr range will be accepted. This is useful when negotiating with some PPP implementations that will not assign an IP number unless their peer requests "`0.0.0.0`".

The next option used "`add default HISADDR`" sets our default route to their IP. This is 'sticky', meaning that if their IP should change, our route will automatically be updated. With "`enable dns`" we are telling our ISP to authenticate our nameserver addresses. Do NOT do this if you are running a local DNS, as ppp will simply circumvent its use by entering some nameserver lines in `/etc/resolv.conf`.

Instead of traditional login methods, many ISPs now use either CHAP or PAP authentication. If this is the case, our configuration will look slightly different:

```
myisp:
  set phone 1234567
  set authname ppp
  set authkey ppp
  set login
  set timeout 120
  set ifaddr 10.0.0.1/0 10.0.0.2/0 255.255.255.0 0.0.0.0
  add default HISADDR
  enable dns
```

In the above example, we specify our username (ppp) and password (ppp) using authname and authkey, respectively. There is no need to specify whether CHAP or PAP authentication is used - it will be negotiated automatically. "`set login`" merely specifies to attempt to log in, with the username and password previously specified.

Using `PPP(8)`

Now that we have our *`ppp.conf`* file set up we can start trying to make a connection to our ISP. I will detail some commonly used arguments with ppp:

- `ppp -auto myisp` - This will run ppp, configure your interfaces and connect to your ISP and then go into the background.
- `ppp -ddial myisp` - This is similar to -auto, but if your connection is dropped it will try and reconnect.

If the above fails, try running */usr/sbin/ppp* with no options - it will run `ppp` in interactive mode. The options can be specified one by one to check for error or other problems. Using the setup specified above, `ppp` will log to `/var/log/ppp.log`. That log, as well as the man page, all contain helpful information.

`ppp(8)` extras

In some situations you might want commands executed as your connection is made or dropped. There are two files you can create for just these situations: `/etc/ppp/ppp.linkup` and `/etc/ppp/ppp.linkdown`. Sample configurations can be viewed here:

- `ppp.linkup`
- `ppp.linkdown`

ppp(8) variations

Many ISPs now offer xDSL services, which are faster than traditional dial-up methods. This includes variants such as ADSL and SDSL. Although no physical dialing takes place, connection is still based on the Point to Point Protocol. Examples include:

- PPPoE
- PPPoA
- PPTP

PPPoE/PPPoA

The Point to Point Protocol over Ethernet (PPPoE) is a method for sending PPP packets in Ethernet frames. The Point to Point Protocol over ATM (PPPoA) is typically run on ATM networks, such as those found in the UK and Belgium.

Typically this means you can establish a connection with your ISP using just a standard Ethernet card and Ethernet-based DSL modem (as opposed to a USB-only modem).

If you have a modem which speaks PPPoE/PPPoA, it is possible to configure the modem to do the connecting. Alternatively, if the modem has a 'bridge' mode, it is possible to enable this and have the modem "pass through" the packets to a machine running PPPoE software (see below).

The main software interface to PPPoE/PPPoA on OpenBSD is `pppoe(8)`, which is a userland implementation (in much the same way that we described `ppp(8)`, above). A kernel PPPoE implementation, `pppoe(4)`, has been incorporated into OpenBSD.

PPTP

The Point to Point Tunneling Protocol (PPTP) is a proprietary Microsoft protocol. A pptp client is available which interfaces with `pppd(8)` and is capable of connecting to the PPTP-based Virtual Private Networks (VPN) used by some cable and xDSL providers. `pptp` itself must be installed from packages or ports. Further instructions on setting up and using `pptp` are available in the man page which is installed with the `pptp` package.

6.6 Tuning networking parameters

One goal of OpenBSD is to have the system *Just Work* for the vast majority of our users. Twisting knobs you don't understand is far more likely to break the system than it is to improve its performance. Always start from the default settings, and *only* adjust things you actually see a problem with.

VERY FEW people will need to adjust any networking parameters!

6.6.1 I don't want the kernel to dynamically allocate a certain port

From sysctl(8):

```
To set the list of reserved TCP ports that should not be allocated by the
kernel dynamically:

      # sysctl net.inet.tcp.baddynamic=749,750,751,760,761,871

This can be used to keep daemons from stealing a specific port that an-
other program needs to function.  List elements may be separated by com-
mas and/or whitespace.

It is also possible to add or remove ports from the current list:

      # sysctl net.inet.tcp.baddynamic=+748
      # sysctl net.inet.tcp.baddynamic=-871
```

6.7 Simple NFS usage

NFS, or Network File System, is used to share a filesystem over the network. A few choice man pages to read before trying to setup a NFS server are:

- nfsd(8)
- mountd(8)
- exports(5)

This section will go through the steps for a simple setup of NFS. This example details a server on a LAN, with clients accessing NFS on the LAN. It does not talk about securing NFS. We presume you have already setup packet filtering or other firewalling protection, to prevent outside access. If you are allowing outside access to your NFS server, and you have any kind of sensitive data stored on it, we strongly recommend that you employ IPsec. Otherwise, people can potentially see your NFS traffic. Someone could also pretend to be the IP address which you are allowing into your NFS server. There are several attacks that can result. When properly configured, IPsec protects against these types of attacks.

Setting up an NFS Server

These services must be enabled and running on the server:

- portmap(8)
- mountd(8)

- nfsd(8)

By default each of these is disabled in OpenBSD. Add the following lines to rc.conf.local(8) to enable them:

```
portmap=YES
nfs_server=YES
```

The next step is to configure the list of filesystems that will be made available for clients to mount.

In this example, we have a server with IP address 10.0.0.1. This server will be serving NFS only to clients within its own subnet. All of this is configured in the /etc/exports file. This file lists which filesystems you wish to have accessible via NFS and defines who is able to access them. There are many options that you can use in /etc/exports; it is best that you read the exports(5) man page. For our example server, we've setup an exports file that looks like this:

```
#
# NFS exports Database
# See exports(5) for more information.  Be very careful, misconfiguration
# of this file can result in your filesystems being readable by the world.
/work -alldirs -ro -network=10.0.0 -mask=255.255.255.0
```

This means that the local filesystem /work will be made available via NFS. The -alldirs option specifies that clients will be able to mount at any point under /work as well as /work itself. For example, if there was a directory called /work/monday, clients could mount /work (and have access to all files/directories underneath that directory) or they could mount /work/monday and have access to just the files/directories contained there. The -ro option specifies that clients will only be granted read-only access. The last two arguments specify that only clients within the 10.0.0.0 network using a netmask of 255.255.255.0 will be authorized to mount this filesystem. This is important for some servers that are accessible by different networks.

Another important security note: don't just add a filesystem to /etc/exports without some kind of list of allowed host(s). Without a list of hosts which can mount a particular directory, anyone who can reach your server will be able to mount your NFS exported directories.

Now you can start the server services. You can either reboot (after enabling them as per the instructions above) or run them manually.

```
# /usr/sbin/portmap
# echo -n >/var/db/mountdtab
# /sbin/mountd
# /sbin/nfsd -tun 4
```

The arguments passed to nfsd enable TCP (-t) and UDP (-u) connections and enable 4 instances (-n) of nfsd to run. You should set an appropriate

number of NFS server instances to handle the maximum number of concurrent client requests that you want to service.

You're now ready to mount the exported filesystems from the client(s).

Remember: If you make changes to `/etc/exports` while NFS is already running, you need to make `mountd` aware of this! Just `HUP mountd` and the changes will take affect.

```
# kill -HUP `cat /var/run/mountd.pid`
```

Mounting NFS Filesystems

NFS filesystems can be mounted from a client without needing to enable any services or daemons. They can be mounted just like any other filesystem.

NFS filesystems should be mounted via `mount(8)`, or more specifically, `mount_nfs(8)`. To mount a filesystem `/work` on host `10.0.0.1` to local filesystem `/mnt`, do this (note that you don't need to use an IP address; mount will resolve host names):

```
# mount -t nfs 10.0.0.1:/work /mnt
```

To have that filesystem mounted at boot, add something like this to `/etc/fstab`:

```
10.0.0.1:/work /mnt nfs rw 0 0
```

It is important that you use `0 0` at the end of this line so that your computer does not try to `fsck` the NFS filesystem on boot. The other standard security options, such as `noexec`, `nodev`, and `nosuid`, should also be used where applicable. For example:

```
10.0.0.1:/work /mnt nfs rw,nodev,nosuid 0 0
```

This way, no devices or setuid programs on the NFS server can subvert security measures on the NFS client. If you are not mounting programs which you expect to run on the NFS client, add `noexec` to this list.

When accessing an NFS mount as the root user, the server automatically maps root's access to username "`nobody`" and group "`nobody`". This is important to know when considering file permissions. For example, take a file with these permissions:

```
-rw-------    1 root     wheel           0 Dec 31 03:00 _daily.B20143
```

If this file was on an NFS share and the root user tried to access this file from the NFS client, access would be denied. This is because the server uses the credentials of the user "`nobody`" when root tries to access the file. Since the user nobody doesn't have permissions to access the file, access is denied.

The user and group that root are mapped to are configurable via the `exports(5)` file on the NFS server.

Checking Stats on NFS

One thing to check to ensure NFS is operating properly is that all the daemons have properly registered with RPC. To do this, use `rpcinfo(8)`.

```
$ rpcinfo -p 10.0.0.1
   program vers proto   port
    100000    2   tcp    111  portmapper
    100000    2   udp    111  portmapper
    100005    1   udp    633  mountd
    100005    3   udp    633  mountd
    100005    1   tcp    916  mountd
    100005    3   tcp    916  mountd
    100003    2   udp   2049  nfs
    100003    3   udp   2049  nfs
    100003    2   tcp   2049  nfs
    100003    3   tcp   2049  nfs
```

During normal usage, there are a few other utilities that allow you to see what is happening with NFS. One is `showmount(8)`, which allows you to view what is currently mounted and who is mounting it. There is also `nfsstat(1)` which shows much more verbose statistics. To use `showmount(8)`, try `/usr/bin/showmount -a host`. For example:

```
$ /usr/bin/showmount -a 10.0.0.1
All mount points on 10.0.0.1:
10.0.0.37:/work
```

This output shows that the client `10.0.0.37` has mounted the `/work` export being served from the server at `10.0.0.1`.

6.8 Setting up a network bridge in OpenBSD

A bridge is a link between two or more separate networks. Unlike a router, packets transfer through the bridge "invisibly" – logically, the two network segments appear to be one segment to nodes on either side of the bridge. The bridge will only forward packets that have to pass from one segment to the other, so among other things, they provide an easy way to reduce traffic in a complex network and yet allow any node to access any other node when needed.

Note that because of this "invisible" nature, an interface in a bridge may or may not have an IP address of its own. If it does, the interface has effectively two modes of operation, one as part of a bridge, the other as a normal, stand-alone NIC. If neither interface has an IP address, the bridge will pass network data, but will not be externally maintainable (which can be a feature).

An example of a bridge application

One of my computer racks has a number of older systems, none of which have a built-in 10BASE-TX NIC. While they all have an AUI or AAUI connector, my supply of transceivers is limited to coax. One of the machines on this rack is an OpenBSD-based terminal server which is always on and connected to the high-speed network. Adding a second NIC with a coax port will allow me to use this machine as a bridge to the coax network.

This system has two NICs in it now, an Intel EtherExpress/100 (`fxp0`) and a 3c590-Combo card (`ep0`) for the coax port. `fxp0` is the link to the rest of my network and will thus have an IP address, `ep0` is going to be for bridging only and will have no IP address. Machines attached to the coax segment will communicate as if they were on the rest of my network. So, how do we make this happen?

The file `hostname.fxp0` contains the configuration info for the `fxp0` card. This machine is set up using DHCP, so its file looks like this:

```
$ cat /etc/hostname.fxp0
dhcp NONE NONE NONE
```

No surprises here.

The `ep0` card is a bit different, as you might guess:

```
$ cat /etc/hostname.ep0
up media 10base2
```

Here, we are instructing the system to activate this interface using `ifconfig(8)` and set it to 10BASE-2 (coax). No IP address or similar information needs to be specified for this interface. The options the ep card accepts are detailed in its man page.

Now, we need to set up the bridge. Bridges are initialized by the existence of a file named something like `hostname.bridge0`. Here is an example for my situation here:

```
$ cat /etc/hostname.bridge0
add fxp0
add ep0
up
```

This is saying set up a bridge consisting of the two NICs, `fxp0` and `ep0`, and activate it. Does it matter which order the cards are listed? No, remember a bridge is very symmetrical – packets flow in and out in both directions.

That's it! Reboot, and you now have a functioning bridge.

Filtering on a bridge

While there are certainly uses for a simple bridge like this, it is likely you might want to DO something with the packets as they go through your bridge. As you

might expect, Packet Filter can be used to restrict what traffic goes through your bridge.

Keep in mind, by the nature of a bridge, the same data flows through both interfaces, so you only need to filter on one interface. Your default "Pass all" statements would look something like this:

```
pass in  on ep0  all
pass out on ep0  all
pass in  on fxp0 all
pass out on fxp0 all
```

Now, let's say I wish to filter traffic hitting these old machines, I want only Web and SSH traffic to reach them. In this case, we are going to let all traffic in and out of the `ep0` interface, but filter on the `fxp0` interface, using `keep state` to handle the reply data:

```
# Pass all traffic through ep0
pass in quick on ep0 all
pass out quick on ep0 all

# Block fxp0 traffic
block in  on fxp0 all
block out on fxp0 all

pass in quick on fxp0 proto tcp from any to any port 22, 80       flags S/SA keep state
```

Note that this rule set will prevent anything but incoming HTTP and SSH traffic from reaching either the bridge machine or any of the other nodes "behind" it. Other results could be had by filtering the other interface.

To monitor and control the bridge you have created, use the `ifconfig(8)` command, which can also be used to create a bridge after boot.

Tips on bridging

- It is HIGHLY recommended that you filter on only one interface. While it is possible to filter on both, you really need to understand this very well to do it right.
- By using the *`blocknonip`* option of `ifconfig(8)` or in `hostname.bridge0`, you can prevent non-IP traffic (such as IPX or NETBEUI) from slipping around your filters. This may be important in some situations, but you should be aware that bridges work for all kinds of traffic, not just IP.
- Bridging requires that the NICs be in a "Promiscuous mode" – they listen to ALL network traffic, not just that directed at the interface. This will put a higher load on the processor and bus than one might expect. Some NICs don't work properly in this mode, the TI ThunderLAN chip (`tl(4)`) is an example of a chip that won't work as part of a bridge.

6.9 How do I boot using PXE? (i386, amd64)

The Preboot Execution Environment, or PXE, is a way to boot a computer from the network, rather than from a hard disk, a floppy or a CD-ROM. The technology was originally developed by Intel, but is supported by most major network card and computer manufacturers now. Note that there are several different network boot protocols, PXE is relatively recent. Traditionally, PXE booting is done using ROMs on the NIC or mainboard of the system, but boot floppies are available from various sources that will permit PXE booting, as well. Many ROMs on older NICs support network booting but do NOT support PXE; OpenBSD/i386 or amd64 cannot currently be booted across the network by these.

How does PXE booting work?

First, it is wise to understand how OpenBSD boots on i386 and amd64 platforms. Upon starting the boot process, the PXE-capable NIC broadcasts a DHCP request over the network. The DHCP server will assign the adapter an IP address, and gives it the name of a file to be retrieved from a `tftp(1)` server and executed. This file then conducts the rest of the boot process. For OpenBSD, the file is `pxeboot`, which takes the place of the standard `boot(8)` file. `pxeboot(8)` is then able to load and execute a kernel (such as `bsd` or `bsd.rd`) from the same `tftp(1)` server.

How do I do it?

The first and obvious step is you must have a PXE-boot capable computer or network adapter. Some documentation will indicate all modern NICs and computers are PXE capable, but this is clearly not true – many low cost systems do not include PXE ROMs or use an older network boot protocol. You also need a properly configured DHCP and TFTP server.

Assuming an OpenBSD machine is the source of the boot files (this is NOT required), your DHCP server `dhcpd.conf` file will need to have the following line:

```
filename "pxeboot";
```

to have the DHCP server offer that file to the booting workstation. For example:

```
shared-network LOCAL-NET
        option  domain-name "example.com";
        option  domain-name-servers 192.168.1.3, 192.168.1.5;

        subnet 192.168.1.0 netmask 255.255.255.0
                option routers 192.168.1.1;
                filename "pxeboot";
```

```
range 192.168.1.32 192.168.1.127;
default-lease-time 86400;
max-lease-time 90000;
```

You will also have to activate the `tftpd(8)` daemon. This is typically done through `inetd(8)`. The standard OpenBSD install has a sample line in `inetd.conf` which will do nicely for you:

```
#tftp  dgram   udp  wait  root  /usr/libexec/tftpd   tftpd -s /tftpboot
```

which simply needs to have the '#' character removed and send `inetd(8)` a `-HUP` signal to get it to reload `/etc/inetd.conf`. `tftpd(8)` serves files from a particular directory, in the case of this line, that directory is `/tftpboot`, which we will use for this example. Obviously, this directory needs to be created and populated. Typically, you will have only a few files here for PXE booting:

- `pxeboot`, the PXE boot loader (serving the same function as boot on a disk-based system).
- `bsd.rd`, the install kernel or bsd, a customized kernel.
- `/etc/boot.conf`, a boot configuration file.

Note that `/etc/boot.conf` is only needed if the kernel you wish to boot from is not named `bsd`, or other pxeboot defaults are not as you need them (for example, you wish to use a serial console). You can test your `tftpd(8)` server using a `tftp(1)` client, making sure you can fetch the needed files.

When your DHCP and TFTP servers are running, you are ready to try it. You will have to activate the PXE boot on your system or network card; consult your system documentation. Once you have it set, you should see something similar to the following:

```
Intel UNDI, PXE-2.0 (build 067)
Copyright (C) 1997,1998 Intel Corporation

For Realtek RTL 8139(X) PCI Fast Ethernet Controller v1.00 (990420)

DHCP MAC ADDR: 00 E0 C5 C8 CF E1
CLIENT IP: 192.168.1.76  MASK: 255.255.255.0  DHCP IP: 192.168.1.252
GATEWAY IP: 192.168.1.1
probing: pc0 com0 com1 apm pxe![2.1] mem[540k 28m a20=on]
disk: hd0*
net: mac 00:e0:c5:c8:cf:e1, ip 192.168.1.76, server 192.168.1.252
>> OpenBSD/i386 PXEBOOT 3.16
boot>
```

At this point, you have the standard OpenBSD boot prompt. If you simply type "`bsd.rd`" here, you will then fetch the file `bsd.rd` from the TFTP server.

```
>> OpenBSD/i386 PXEBOOT 3.16
   boot> bsd.rd
   booting tftp:bsd.rd: 4375152+733120 [58+122112+105468]=0x516d04
   entry point at 0x100120

   Copyright (c) 1982, 1986, 1989, 1991, 1993
           The Regents of the University of California.  All rights reserved.

   Copyright (c) 1995-2011 OpenBSD.  All rights reserved.  http://www.OpenBSD.org

   OpenBSD 5.0 (RAMDISK_CD) #36: Wed Aug 17 10:27:31 MDT 2011
     ...
```

The `bsd.rd` install kernel will now boot.

Can I boot other kinds of kernels using PXE other than `bsd.rd`?

Yes, although with the tools currently in OpenBSD, PXE booting is primarily intended for installing the OS.

6.10 The Common Address Redundancy Protocol (CARP)

6.10.1 What is CARP and how does it work?

CARP is a tool to help achieve system redundancy, by having multiple computers creating a single, virtual network interface between them, so that if any machine fails, another can respond instead, and/or allowing a degree of load sharing between systems. CARP is an improvement over the Virtual Router Redundancy Protocol (VRRP) standard. It was developed after VRRP was deemed to be not free enough because of a possibly-overlapping Cisco patent. For more information on CARP's origins and the legal issues surrounding VRRP, please visit this page.

To avoid legal conflicts, Ryan McBride (with help from Michael Shalayeff, Marco Pfatschbacher and Markus Friedl) designed CARP to be fundamentally different. The inclusion of cryptography is the most prominent change, but still only one of many.

How it works: CARP is a multicast protocol. It groups several physical computers together under one or more virtual addresses. Of these, one system is the master and responds to all packets destined for the group, the other systems act as hot spares. No matter what the IP and MAC address of the

local physical interface, packets sent to the CARP address are returned with the CARP information.

At configurable intervals, the master advertises its operation on IP protocol number 112. If the master goes offline, the other systems in the CARP group begin to advertise. The host that's able to advertise most frequently becomes the new master. When the main system comes back up, it becomes a back up host by default, although if it's more desirable for one host to be master whenever possible (e.g. one host is a fast Sun Fire V120 and the others are comparatively slow SPARCstation IPCs), you can so configure them.

While highly redundant and fault-tolerant hardware minimizes the need for CARP, it doesn't erase it. There's no hardware fault tolerance that's capable of helping if someone knocks out a power cord, or if your system administrator types `reboot` in the wrong window. CARP also makes it easier to make the patch and reboot cycle transparent to users, and easier to test a software or hardware upgrade–if it doesn't work, you can fall back to your spare until fixed.

There are, however, situations in which CARP won't help. CARP's design does require that the members of a group be on the same physical subnet with a static IP address, although with the introduction of the carpdev directive, there is no more need for IP addresses on the physical interfaces. Similarly, services that require a constant connection to the server (such as SSH or IRC) will not be transparently transferred to the other system–though in this case, CARP can help with minimizing downtime. CARP by itself does not synchronize data between applications, this has to be done through "alternative channels" such as `pfsync(4)` (for redundant filtering), manually duplicating data between boxes with `rsync`, or whatever is appropriate for your application.

6.10.2 Configuration

CARP's controls are located in two places: `sysctl(8)` and `ifconfig(8)`. Let's look at the sysctls first.

The first sysctl, `net.inet.carp.allow`, defines whether the host handles CARP packets at all. Clearly, this is necessary to use CARP. This sysctl is enabled by default.

The second, `net.inet.carp.log`, logs CARP state changes, bad packets and other errors. Set to log state changes by default.

Third, `net.inet.carp.preempt` enables natural selection among CARP hosts. The most fit for the job (that is to say, able to advertise most frequently) will become master. Disabled by default, meaning a system that is not a master will not attempt to (re)gain master status.

All these sysctl variables are documented in `sysctl(3)`.

For the remainder of CARP's configuration, we rely on `ifconfig(8)`. The CARP-specific commands `advbase` and `advskew` deal with the interval between CARP advertisements. The formula (in seconds) is `advskew` divided by 256, then added to advbase. `advbase` can be used to decrease network traffic or allow longer latency before a backup host takes over; `advskew` lets you control which host will be master without much delaying failover (should that be required).

Next, `pass` sets a password, and `vhid` sets the virtual host identifier number of the CARP group. You need to assign a unique number for each CARP group, even if (for load balancing purposes) they share the same IP address. CARP is limited to 255 groups.

Finally, `carpdev` specifies which physical interface belongs to this particular CARP group. By default, whichever interface has an IP address in the same subnet assigned to the CARP interface will be used.

Let's put all these settings together in a basic configuration. Let's say you're deploying two identically configured Web servers, *rachael* (192.168.0.5) and *pris* (`192.168.0.6`), to replace an older system that was at `192.168.0.7`. The commands:

```
rachael# ifconfig carp0 create
rachael# ifconfig carp0 vhid 1 pass tyrell carpdev fxp0      192.168.0.7 netmask 255.255.255.0
```

create the `carp0` interface and give it a vhid of 1, a password of *tyrell*, and the IP address `192.168.0.7` with mask `255.255.255.0`. Assign `fxp0` as the member interface. To make it permanent across reboots, you can create an `/etc/hostname.carp0` file that looks like this:

```
inet 192.168.0.7 255.255.255.0 192.168.0.255 vhid 1 pass tyrell carpdev fxp0
```

Note that the broadcast address is specified in that line, in addition to the vhid and the password. Failing to do this is a common cause of errors, as it is needed as a place holder.

Do the same on *pris*. Whichever system brings the CARP interface up first will be master (assuming that preempt is disabled; the opposite is true when preempt is enabled).

But let's say you're not deploying from scratch. *Rachael* was already in place at the address `192.168.0.7`. How do you work around that? Fortunately, CARP can deal with this situation. You simply assign the address to the CARP interface and leave the physical interface specified by the '`carpdev`' keyword without an IP address. However, it tends to be cleaner to have an IP for each system–it makes individual monitoring and access much simpler.

Let's add another layer of complexity; we want *rachael* to stay master when possible. There are several reasons we might want this: hardware differences, simple prejudice, "if this system isn't master, there's a problem," or knowing the default master without doing scripting to parse and email the output of ifconfig.

On *rachael*, we'll use the sysctl we created above, then edit `/etc/sysctl.conf` to make it permanent.

```
rachael# sysctl net.inet.carp.preempt=1
```

We'll do configuration on *pris*, too:

```
pris# ifconfig carp0 advskew 100
```

This slightly delays *pris's* advertisements, meaning `rachael` will be master when alive.

Note that if you are using PF on a CARP'd computer, you must pass "`proto carp`" on all involved interfaces, with a line similar to:

```
pass on fxp0 proto carp keep state
```

6.10.3 Load balancing

Flash forward a few months. Our company of the previous example has grown to the point where a single internal Web server is just barely managing the load. What to do? CARP to the rescue. It's time to try load balancing. Create a new CARP interface and group on *rachael*:

```
rachael# ifconfig carp1 create
rachael# ifconfig carp1 vhid 2 advskew 100 pass bryant carpdev fxp0      192.168.0.7 ne
```

On *pris*, we'll create the new group and interface as well, then set the "`preempt`" sysctl:

```
pris# ifconfig carp1 create
pris# ifconfig carp1 vhid 2 pass bryant carpdev fxp0      192.168.0.7 netmask 255.255.2
pris# sysctl net.inet.carp.preempt=1
```

Now we have two CARP groups with the same IP address. Each group is skewed toward a different host, which means *rachael* will stay master of the original group, but *pris* will take over the new one.

While these examples are for a two-machine cluster, the same principles apply to more systems. Please note, however, that it's not expected that you will achieve perfect 50/50 distribution between the two machines–CARP uses a hash of the originating IP address to determine which system handles the request, rather than by load.

6.10.4 More Information on CARP

- `carp(4)`
- `ifconfig(8)`
- `sysctl(8)`
- `sysctl(3)`
- Firewall Failover with pfsync and CARP by Ryan McBride

6.11 Using OpenNTPD

Accurate time is important for many computer applications. However, many people have noticed that their 5*watchcankeepbettertimethantheir*2000 computer. In addition to knowing what time it is, it is also often important to synchronize computers so that they all agree on what time it is. For some time, ntp.org has produced a Network Time Protocol (RFC1305, RFC2030) application, available through ports, which can be used to synchronize clocks on computers over the Internet. However, it is a nontrivial program to set up, difficult code to audit, and has a large memory requirement. In short, it fills an important role for some people, but it is far from a solution for all.

OpenNTPD is an attempt to resolve some of these problems, making a trivial-to-administer, safe and simple NTP compatible way to have accurate time on your computer. OpenBSD's `ntpd(8)` is controlled with an easy to understand configuration file, `/etc/ntpd.conf`.

Simply activating `ntpd(8)` through `rc.conf.local` will result in your computer's clock slowly moving towards, then keeping itself synchronized to, the `pool.ntp.org` servers, a collection of publicly available time servers. Once your clock is accurately set, `ntpd` will hold it at a high degree of accuracy, however, if your clock is more than a few minutes off, it is highly recommended that you bring it to close to accurate initially, as it may take days or weeks to bring a very-off clock to sync. You can do this using the "`-s`" option of `ntpd(8)` or any other way to accurately set your system clock.

6.11.1 "But OpenNTPD isn't as accurate as the ntp.org daemon!"

That may be true. That is not OpenNTPD's design goal, it is intended to be free, simple, reliable and secure. If you really need microsecond precision more than the benefits of OpenNTPD, feel free to use ntp.org's ntpd, as it will remain available through ports and packages. There is no plan or desire to have OpenNTPD bloated with every imaginable feature.

6.11.2 "Someone has claimed that OpenNTPD is 'harmful'!"

Some people have not understood the goals of OpenNTPD – a simple, secure and easy to maintain way to keep your computer's clock accurate. If accurate time keeping is important, a number of users have reported better results from OpenNTPD than from ntp.org's ntpd. If security is important, OpenNTPD's code is much more readable (and thus, auditable) and was written using native OpenBSD function calls like `strlcpy`, rather than more portable functions like `strcpy`, and written to be secure from the beginning, not "made secure later". If having more people using time synchronization is valuable, OpenNTPD makes it much easier for larger numbers of people to use it. If this is "harmful", we are all for it.

There are applications where the ntp.org ntpd is more appropriate; however it is felt that for a large majority of the users, OpenNTPD is more than sufficient.

A more complete response to this by one of the maintainers of OpenNTPD can be read here.

6.11.3 Why can't my other machines synchronize to OpenNTPD?

`ntpd(8)` does not listen on any address by default. So in order to use it as a server, you have to uncomment the "`#listen on *`" line in `/etc/ntpd.conf` and restart the `ntpd(8)` daemon. Of course, if you wish it to listen on a particular IP address rather than all available addresses and interfaces, replace the "`*`" with the desired address.

When you have `ntpd(8)` listening, it may happen that other machines still can't synchronize to it! A freshly started `ntpd(8)` daemon (for example, if you just restarted it after modifying `ntpd.conf`) refuses to serve time information to other clients until it adjusts its own clock to a reasonable level of stability first. When `ntpd(8)` considers its own time information stable, it announces it by a "`clock now synced`" message in `/var/log/daemon`. Even if the system clock is pretty accurate in the beginning, it can take up to 10 minutes to get in sync, and hours or days if the clock is not accurately set at the start.

6.12 What are my wireless networking options?

OpenBSD has support for a number of wireless chipsets:

- `acx(4)` TI ACX100/ACX111. (NFF) (AP)
- `an(4)` Aironet Communications 4500/4800.
- `ath(4)` driver for Atheros 802.11a/b/g. (AP)
- `athn(4)` driver for Atheros 80211/a/g/n devices.
- `atu(4)` Atmel AT76C50x USB 802.11b
- `atw(4)` ADMtek ADM8211.
- `awi(4)` AMD 802.11 PCnet Mobile.
- `bwi(4)` Broadcom AirForce 802.11b/g
- `cnwi(4)` Xircom CreditCard Netwave
- `ipw(4)` Intel PRO/Wireless 2100 802.11b. (NFF)
- `iwi(4)` Intel PRO/Wireless 2200BG/2225BG/2915ABG 802.11a/b/g. (NFF)
- `iwn(4)` Intel WiFi Link 4965/5100/5300 802.11a/b/g/Draft-N wireless.

- `malo(4)` Marvell Libertas 802.11b/g
- `pgt(4)` Conexant/Intersil Prism GT Full-MAC 802.11a/b/g
- `ral(4)` and ural(4) [USB] Ralink Technology RT25x0 802.11a/b/g. (AP)
- `ray(4)` Raytheon Raylink/WebGear Aviator 802.11FH
- `rtw(4)` Realtek 8180 802.11b. (AP)
- `rum(4)` Ralink Technology RT2501USB. (AP)
- `run(4)` Ralink Technology USB 802.11a/b/g/Draft-N
- `uath(4)` Atheros USB 802.11a/b/g
- `upgt(4)` Conexant/Intersil PrismGT SoftMAC USB 802.11b/g
- `urtw(4)` Realtek RTL8187L USB 802.11b/g
- `wi(4)` Prism2/2.5/3. (AP)
- `wpi(4)` Intel PRO/Wireless 3945ABG. (NFF)
- `zyd(4)` ZyDAS ZD1211/ZD1211B USB 802.11b/g

Chapter 7

Keyboard and Display Controls

7.1 How do I remap the keyboard? (*wscons*)

Most modern OpenBSD platforms use the console driver.

With `wscons(4)` consoles, most options can be controlled using the `wsconsctl(8)` utility. For example, to change keymappings with `wsconsctl(8)` one would execute the following:

```
# 'wsconsctl keyboard.encoding=uk
```

In the next example, we will remap "Caps Lock" to be "Control L" (the left control key):

```
# wsconsctl keyboard.map+="keysym Caps_Lock = Control_L''
```

7.2 Is there console mouse support in OpenBSD?

For the alpha, amd64 and i386 platforms, OpenBSD provides `wsmoused(8)`, a port of FreeBSD's `moused(8)`. It can be enabled automatically at startup by copying and editing the appropriate line from `rc.conf` to `rc.conf.local(8)`.

7.3 Accessing the Console Scrollback Buffer (*amd64, i386, some Alpha*)

On a few platforms, OpenBSD provides a console scrollback buffer. This allows you to see information that has already scrolled past your screen. To move up and down in the buffer, simply use the key combinations Shift ⇑ + Page ↑ and Shift ⇑ + Page ↓ .

The default scrollback buffer, or the number of pages that you can move up and view, is 8. This is a feature of the `vga(4)` driver, so it will not work without a VGA card on any platform (many Alpha systems have TGA video).

Due to space limitations, the install kernels do not provide the scrollback function. Switching consoles will clear the scrollback buffer.

7.4 How do I switch consoles? (*amd64, i386, Zaurus, some Alpha*)

On amd64, i386 and Alpha systems with vga(4) cards, OpenBSD provides six virtual terminals by default, `/dev/ttyC0` through `/dev/ttyC5`. `ttyC4` is reserved for use by the X Window system, leaving five text consoles. You can switch between them using Ctrl + Alt + F1 , Ctrl + Alt + F2 , Ctrl + Alt + F3 , Ctrl + Alt + F4 and Ctrl + Alt + F6 .

The X environment uses `ttyC4`, Ctrl + Alt + F5 . When using X, the Ctrl + Alt + F*n* keys will take you to the text screens; Ctrl + Alt + F5 will take you back to the graphical environment.

If you wish to have more than the default number of virtual consoles, use the wsconscfg(8) command to create screens for `ttyC6`, `ttyC7` and above. For example:

```
wsconscfg -t 80x25 6
```

will create a virtual terminal for `ttyC6`, accessed by Ctrl + Alt + F7 . Don't forget to add this command to your `rc.local(8)` file if you want the extra screen the next time you boot the computer.

Note that you will not get a "login:" prompt on the newly-created virtual console unless you set it to "`on`" in `/etc/ttys(5)`, and either reboot or send `init(8)` a `HUP` signal using `kill(1)`.

On the Zaurus, two virtual terminals (`/dev/ttyC0` and `/dev/ttyC1`) are available by default, accessible with Alt Calendar and Alt + Address (The Alt key is the one right of the left Ctrl key).

7.5 How do I use a console resolution of 80x50? (*amd64, i386, some Alpha*)

amd64, i386, and VGA Alpha users normally get a console screen of 25 lines of 80 characters. However, many VGA video cards are capable of displaying a higher text resolution of 50 lines of 80 characters.

First, a font that supports the desired resolution must be loaded using the `wsfontload(8)` command. The standard 80x25 text screen uses 8x16 pixel fonts; to double the number of lines we will have to use 8x8 pixel fonts.

After that, we will have to delete and recreate a virtual console at the desired screen resolution, using the `wsconscfg(8)` command.

This can be done automatically at boot by adding the following lines to the end of your `rc.local(8)` file:

```
wsfontload -h 8 -e ibm /usr/share/misc/pcvtfonts/vt220l.808
wsconscfg -dF 5
wsconscfg -t 80x50 5
```

As with any modification to your system configuration, it is recommended you spend some time with the man pages to understand what these commands do.

The first line above loads the 8x8 font. The second line deletes screen 5 (which would be accessed by Ctrl + Alt + F6). The third line creates a new screen 5 with 50 lines of 80 characters each. If you do this, you will see your primary screen, and the other three default virtual consoles, come up in the standard 80x25 mode, but a new screen 5 at 80x50 accessible through Ctrl + Alt + F6 .

Remember that Ctrl + Alt + F6 is screen 0 (`ttyC0`). If you wish to alter other screens, simply repeat the delete and add screen steps for whichever screens you want running at the 80x50 resolution.

You should avoid changing screen 4 (`ttyC4`, Ctrl + Alt + F5), which is used by X as a graphical screen. It is also not possible to change the resolution of the primary console device (i.e., `ttyC0`).

As one might expect, all these commands can also be entered at the command prompt, as root, or (better) using `sudo(8)`.

Note: this will not work on all video cards. Unfortunately, not all video cards support the uploaded fonts that `wscons(4)` requires to achieve the 80x50 text mode. In these cases, you might wish to consider running X.

7.6 How do I use a serial console?

There are many reasons you may wish to use a serial console for your OpenBSD system:

- Recording console output (for documentation).
- Remote management.
- Easier maintenance of a large quantity of machines
- Providing a useful dmesg from machines which might otherwise be difficult to get one from.
- Providing an accurate "trace" and "ps" output if your system crashes so developers can have a chance to fix the problem.

OpenBSD supports serial console on most platforms, however details vary greatly between platforms.

Note that serial interfacing is NOT a trivial task – you will often need unusual cables, and ports are not standardized between machines, in some cases, not even consistent on one machine. It is assumed you know how to select the appropriate cable to go between your computer and the device acting as your serial terminal. A full tutorial on serial interfacing is beyond the scope of this article, however, we offer one hint: just because the ends plug in doesn't mean it will work.

/etc/ttys change

There are two parts to getting a functional serial console on an OpenBSD system. First, you must have OpenBSD use your serial port as a console for status and single user mode. This part is very platform dependent. Second, you must enable the serial port to be used as an interactive terminal, so a user can log into it when running multi-user. This part is fairly similar between platforms, and is detailed here.

Terminal sessions are controlled by the */etc/ttys* file. Before OpenBSD will give you a "`login:`" prompt at a device, it has to be enabled in `/etc/ttys`, after all, there are other uses for a serial port other than for a terminal. In

platforms which typically have an attached keyboard and screen as a console, the serial terminal is typically disabled by default. We'll use the i386 platform as an example. In this case, you must edit the line that reads:

```
tty00   "/usr/libexec/getty std.9600"   unknown off
```

to read something like:

```
tty00   "/usr/libexec/getty std.9600"   vt220   on secure
```

Here, `tty00` is the serial port we are using as a console. `vt220` is the `termcap(5)` entry that matches YOUR terminal (other likely options might include `vt100`, `xterm`, etc.). The "`on`" activates the getty for that serial port so that a "`login:`" prompt will be presented, the "`secure`" permits a root (uid 0) login at this console (which may or may not be what you desire), and the "`9600`" is the terminal baud rate. Resist the urge to crank the baud rate up to the maximum your hardware can support, as you are more likely to create problems than benefit. Most systems have a "default" speed (supported by default by the boot ROM and/or the boot loader, often 9600), use this unless you have real reason to use something different.

Note that you can use a serial console for install without doing this step, as the system is running in single user mode, and not using `getty` for login.

On some platforms and some configurations, you must bring the system up in single user mode to make this change if a serial console is all you have available.

amd64 and i386

To direct the boot process to use the serial port as a console, create or edit your */etc/boot.conf* file to include the line:

```
set tty com0
```

to use the first serial port as your console. The default baud rate is 9600bps, this can be changed with a */etc/boot.conf* line using the stty option. This file is put on your boot drive, which could also be your install floppy, or the command can be entered at the boot> prompt from the OpenBSD second-stage boot loader for a one-time (or first time) serial console usage.

amd64 and i386 notes:

- OpenBSD numbers the serial ports starting at *tty00*, DOS/Windows labels them starting at *COM1*. So, keep in mind *tty02* is *COM3*, not *COM2*.
- Some systems may be able to operate without a video card in the machine, but certainly not all – many systems consider this an error condition. Some machines will even refuse to work easily without a keyboard attached.

- Some systems are capable of redirecting all BIOS keyboard and screen activity to a serial port through a configuration option, so the machine can be completely maintained through the serial port. Your results may vary – when using this feature, some BIOSs may prevent the bootloader from seeing the serial port, and thus, the kernel will not be told to use it. Some BIOSs have an option to "Continue Console Redirection after POST" (Power On Self Test), this should be set to "OFF", so the boot loader and the kernel can handle their own console. Unfortunately, this feature is not universal.

- PC compatible computers are not designed to be run from a serial console, unlike some other platforms. Even those systems that support a serial console usually have it as a BIOS configuration option – and should the configuration information get corrupted, you will find the system looking for a standard monitor and keyboard again. You generally must have some way to get a monitor and keyboard to your amd64 and i386 systems in an emergency.

- You will need to edit `/etc/ttys` as above.

SPARC and UltraSPARC

These machines are designed to be completely maintainable with a serial console. Simply remove the keyboard from the machine, and the system will run serial.

SPARC and UltraSPARC notes

- The serial ports on a SPARC are labeled *ttya*, *ttyb*, etc.

- Unlike some other platforms, it is not necessary to make any changes to */etc/ttys* to use a serial console.

- The SPARC/UltraSPARC systems interpret a BREAK signal on the console port to be the same as a STOP-A command, and kicks the system back to the Forth prompt, stopping any application and operating system at that point. This is handy when desired, but unfortunately, some serial terminals at power-down and some RS-232 switching devices send something the computer interprets as a break signal, halting the machine. Test before you go into production.

- If you have a keyboard and monitor attached, you can still force the serial console to be used instead by using the following commands at the `ok` prompt:

```
ok setenv input-device ttya
ok setenv output-device ttya
ok reset
```

If the keyboard and monitor (`ttyC0`) are active in */etc/ttys* (above), you can use the keyboard and monitor in X.

MacPPC

The MacPPC machines are configured for a serial console through OpenFirmware. Use the commands:

```
ok setenv output-device scca
ok setenv input-device scca
ok reset-all
```

Set your serial console to 57600bps, 8N1.

MacPPC notes

- Unfortunately, serial console is not directly possible on most MacPPCs. While most of these machines do have serial hardware, it isn't accessible outside the machine. Fortunately, a few companies offer add-on devices for several Macintosh models which will make this port available for use as a serial console (or other uses). Use your favorite search engine and look for "Macintosh internal serial port".

- You will have to change *tty00* in */etc/ttys* to on and set the speed to 57600 instead of the default of 9600 as detailed above in single user mode before booting multi-user and having the serial console functional.

Mac68k

Serial console is selected in the *Booter* program, under the "`Options`" pull-down menu, then "`Serial Ports`". Check the "Serial Console" button, then choose the Modem or Printer port. You will need a Macintosh modem or printer cable to attach to the Mac's serial ports. If you wish to have this as default, tell the Booter program to save your options.

Mac68k Notes

- The modem port is *tty00*, the printer port is *tty01*.

- The Mac68k doesn't turn on its serial port until called upon, so your breakout box may not show any signals on the Mac's serial port until the OpenBSD boot process has started.

- You will have to enable the port (*tty00* or *tty01*) as indicated above.

7.7 How do I blank my console? (wscons)

If you wish to blank your console after a period of inactivity without using X, you can alter the following `wscons(4)` variables:

- **`display.vblank`** set to `on` will disable the vertical sync pulse, which will cause many monitors to go into an "energy saver" mode. This will require more time to bring the screen back on, but will reduce energy consumption and heat production of newer monitors. When set to off, the display will blank, but the monitor will still be receiving the normal horizontal and vertical sync pulses, so the unblanking will be almost instant.
- **`display.screen_off`** determines the blanking time in thousandths of a second, i.e., 60000 would set the timeout to one minute.
- **`display.kbdact`** determines if keyboard activity will restore the blanked screen. Usually, this is desirable.
- **`display.outact`** determines if screen output will restore the blanked screen.

You can set these variables at the command line using the `wsconsctl(8)` command:

```
# wsconsctl display.screen_off=60000
    display.screen_off -> 60000
```

or set them permanently by editing */etc/wsconsctl.conf* so these changes take place at next boot:

```
display.vblank=on               # enable vertical sync blank
display.screen_off=600000       # set screen blank timeout to 10 minutes
display.kbdact=on               # Restore screen on keyboard input
display.outact=off              # Restore screen on display output
```

The blanker is activated when either `display.kbdact` or `display.outact` is set to "`on`".

7.8 EVERYTHING I TYPE AT THE LOGIN PROMPT IS IN CAPS!

This is a feature, not a bug, actually.

Virtually all Unix commands and user names are entered using all lowercase. However, some very old terminals were only capable of uppercase characters, making them difficult, if not impossible, to use with Unix. As a workaround, if you entered your user name in all uppercase, `getty(8)` would assume your terminal was "lowercase challenged", and simply interpret everything you type as

lowercase, while echoing it as uppercase. If you have a mixed-case or uppercase password, this will make login impossible.

Hitting Ctrl + D at the login prompt will cause `getty(8)` to terminate, and `init(8)` will relaunch a new one, which will accept uppercase and lowercase properly.

7.9 What is tmux?

For those familiar with the "screen" program, provided as a package, or window(1) which used to be in base, it may be easiest to answer this by saying that `tmux(1)` performs many of the functions as screen and window, with many additional features.

For those not familiar with these programs, tmux is a *terminal multiplexer*. This is a program which allows a number of other processes to share the same screen for input and output. In tmux, such a collection of programs is known as a *session*, with each program contained in a tmux *window*.

In addition to sharing the terminal, tmux lets you detach a session and its windows from the screen, leave them continue running in the background, and later reattach them to the same or to a different screen. A session may be detached manually or through an unexpected event such as network disconnection, in either case the programs survive and continue running as normal.

tmux also has many other features such as splitting a single window into multiple sections (known as panes), a history of text printed in each window, copying and pasting text between windows, configurable key bindings, and terminal locking. Take a look at the `tmux(1)` man page for further information.

How do I use tmux?

The first step is to run tmux:

```
$ tmux
```

This starts a new tmux with a single new session (called "0") and creates a client displaying it on screen. Most of the screen will show a window containing a shell prompt, and you will notice the last line is occupied by a status line. This shows the name of the session in square brackets on the left, the window title (normally empty for shells) and the time on the right, and a summary of the current open windows in the middle. In your new session, the currently open windows will contain one entry, for example:

```
0:ksh*
```

A brief aside regarding terminals: on OpenBSD, applications are made aware of the capabilities of the terminal by the `TERM` environment variable. This is set to the name of an entry in the `terminfo(5)` database and tells programs that the terminal, for example, supports colour, or has the ability to insert lines,

or many other things. An important thing to note is that the "`xterm`" entry in the database does not include colour, so tmux will not use colour in xterm by default, `TERM` should be set to "`xterm-xfree86`" instead if colour is desired (the "`XTerm*termName`" X resource may be set in `.Xdefaults` to use this for all xterms). It is also important that `TERM` in shells started inside tmux is set to "screen" or programs run from them may not display correctly - tmux will set this itself, but care should be taken not to override it in shell startup files.

Returning to the status line, the number "0" is the window index, "`ksh`" the name of the window, and the "`*`" indicates this is the current window displayed above the status line. Any typing is passed on to the shell and any output displayed. For example, if you start "`top`":

```
$ top
```

It will run as normal, occupying the part of the screen above the status line. You may also notice that the window name in the status line has changed from "`ksh`" to "`top`" - tmux renames windows to reflect the program currently running in them.

Now, let's say you want to detach tmux from the screen and return to the original shell from which you started it. A tmux session may be detached by first pressing the Ctrl and b keys together, and then the d key. The Ctrl + b key combination (shortened in tmux and its man page to "`C-b`") is known as the prefix key and is used to tell tmux that the next key pressed is an instruction that it should perform some action, rather than passing the key through to the program in the window.

After pressing Ctrl + b d and returning to the shell prompt, reattach the tmux session using the "`attach`" command:

```
$ tmux attach
```

The tmux session will reappear, with the status line and "`top`" still happily running. (If you instead run tmux again without arguments, a second session will be created, named "`1`".)

Next, let's create a second window. This is done using the "`c`" key: press the prefix key, Ctrl + b , then the c key. A new window will be created and again a shell prompt displayed on screen. The status line will be updated to show the new window:

```
0:top- 1:ksh*
```

The "`-`" after "`top`" shows the previously current window (the last window). Pressing Ctrl + b c again creates another new shell:

```
0:top  1:ksh- 2:ksh*
```

There are several commands for moving between windows. From window 2, you can move the previous window, number 1, by typing Ctrl + b p .

Ctrl + b n moves to the next window: in this case, there is no window 3 so the current window wraps to window 0. You can also press Ctrl + b w to get an interactive menu of open windows, Ctrl + b l to move to the last window (the one marked with "-"), or Ctrl + b 0 to move to window 0, Ctrl + b 1 for window 1 and so on up to Ctrl + b 9 for window 9. So, to get back to "`top`" in window 0, you can press Ctrl + b 0 to go directly to window 0, Ctrl + b p twice to move via window 1, Ctrl + b n to wrap from window 2 to window 0, or press Ctrl + b w and select window 0 from the list.

Sometimes you may want to create a window running a program directly, without using a shell first. This can be done from the tmux command prompt. Pressing the Ctrl + b : key sequence changes the status line to display a "`:`" prompt at which commands may be entered. All the tmux commands are documented in the man page. In this case the "`new-window`" command is needed. Each command has a shorthand alias which may be used instead of typing the full name, for "`new-window`" this is "`neww`". So, to create a new window running `tetris(6)`, type:

```
neww tetris
```

The new window will close when tetris exits, or may be forcibly killed using the Ctrl + b & key binding. This will first prompt for confirmation and if given, close the window and terminate the program running in it.

Another common requirement is renaming a window. This can be done with the Ctrl + b , key binding. The status line will change to display a "`(rename-window)`" prompt at which the new name may be entered. Renaming a window turns off automatic renaming for that window, to reenable that feature, Ctrl + b : to get to the command prompt and enter the following (more on what this means is in the next section):

```
setw -u automatic-rename
```

One other important key is worth remembering: Ctrl + b ?. This will show a list of all the tmux keys and the commands they execute. For example, Ctrl + b ? shows that the c key is bound to the "`new-window`" command and the n key to the "`next-window`" command.

Configuring tmux

Many users want to customise the way tmux looks or behaves. This is done through the configuration file, *`/.tmux.conf`*. This file is a list of tmux commands which are executed when tmux is initially started, before the first session is created. All tmux commands are documented in the man page, but a few common examples you might want to put in your configuration file are discussed below.

The most common requirement is setting options. There are two types of option in tmux: *session options* and *window options*. Session options control

the behaviour of a session and window options of an individual window. For both there is a set of global options. When tmux comes to decide on an options value for a particular session or window, it looks first at the options local to that window; if the option has not been set, the global option value is used.

Session options are set with the "set-option" (alias "set") command and window options with the "`set-window-option`" command (alias "`setw`"). To set a global option, use the "`-g`" flag, if this is left out the option is set for the current window or session. These commands also accept a few other flags, such as "`-u`" to unset a local option and allow a window or session to inherit the global option again.

In the configuration file, it is usual to set global options. Let's look at some examples customising the status line:

```
set -g status-bg blue
set -g status-right '#(sysctl vm.loadavg)'
setw -g window-status-current-attr underscore
```

Putting these three commands in `.tmux.conf` and restarting tmux changes the status line background to blue, puts the current load average on the right side and underlines the current window. The status line may be turned off entirely with:

```
set -g status off
```

There are a large number of other options; another handy one is changing to `vi(1)`-style key bindings at the command prompt and in the window list and other tmux interactive modes:

```
set -g status-keys vi
setw -g mode-keys vi
```

The current options and their values may be listed with the "`show-options`" and "`show-window-options`" commands. Like the set commands these accept "`-g`" to show the global options.

Another common task for the configuration file is adding or modifying tmux command key bindings, that is the commands that are executed after you press Ctrl + b with another key. These are added or changed with the "`bind-key`" command (alias "`bind`") and removed with the "`unbind-key`" command (alias "`unbind`"). Two examples of using "`bind-key`" are:

```
bind C-d detach
bind / neww 'exec top'
```

The first line creates a binding for Ctrl + b Ctrl + d to detach tmux, the same as the default Ctrl + b d, and the second binds Ctrl + b / to create a new window running top.

Many people like to use a different prefix key than Ctrl + b. This can be achieved using both the "`set-option`" and key binding commands to alter

the prefix key option and change so that pressing the prefix twice continues to pass the same key through to the window. To change the prefix key to Ctrl + a :

```
set -g prefix C-a
unbind C-b
bind C-a send-prefix
```

The final useful thing to do in the configuration file is to create an initial set of windows when tmux starts. This is slightly more complicated than the previous examples. In tmux, a session cannot have no windows, and you cannot create windows without a session. So, to have windows created by the configuration file you must first create a session to contain them. For example (note that "`new`" is the alias for the "`new-session`" command):

```
new -d 'exec top'
neww -d
neww -d
```

These commands create a new session with "`top`" running in the first window, then create two additional windows. The "`-d`" flags instruct tmux not to try to attach the new session or to make the new windows the current window. Before putting these lines into `.tmux.conf`, there is one other issue. When executed without arguments, tmux runs the "`new-session`" command, so when starting tmux with "`tmux`" from the shell, the configuration file tells tmux to create one session, then the command from the shell tells it to create another, so you end up with two sessions. To avoid this, tmux should be started with "`tmux attach`" when creating a session from the configuration file - this means it will create the session from `.tmux.conf` then immediately attach to it without creating a second session.

7.9.1 Advanced tmux usage

This section briefly covers some of the more advanced features available in tmux. See the man page for more information.

In tmux, you can copy and paste text between windows. This is done by copying the text in *copy mode* and then pasting it with the paste command. To enter copy mode, press Ctrl + b [. In copy mode (with the "`mode-keys`" window option set to `emacs`, for `vi` keys see the man page) the arrow keys may be used to position the cursor, Ctrl + Space starts the selection and Ctrl + w copies. You can also use Page ↑ , Page ↓ , Ctrl + a and Ctrl + e to move the cursor around. Press q or Esc to exit copy mode. After that, Ctrl + b] will paste the copied text into the current window as if you had typed it directly.

tmux is quite scriptable, and most commands that may be entered from the command prompt or bound to a key may be executed from the shell. Almost all tmux commands accept an optional "`-t`" argument which specifies the session or window on which to act. For example, this command:

```
$ tmux kill-window -t0:1
```

will kill window 1 in session 0. And:

```
$ tmux new-window -tmysession
```

creates a new window in the session named "`mysession`". Many commands accept other arguments, for example the "`new-window`" command accepts a "`-n`" option to give the name of the new window, and "`new-session`" accepts several arguments to specify the attributes of the initial window created with the session. These arguments may naturally be used when a command is bound to a key or executed from the command prompt as well.

Another useful feature is the ability to split a single window into several sections, called panes. You can split a window vertically (top to bottom) with the Ctrl + b " key combination. A pane can be resized up or down with Ctrl + b Alt + ↑ and Ctrl + b Alt + ↓ and the active pane changed with Ctrl + b o . In addition, a window split in that way may be changed into one of a number of fixed layouts, these are cycled through with Ctrl + b Space but panes in one of these layouts may not be resized. In -current, splitting has been extended to support horizontal splitting Ctrl + b %) and the fixed layouts changed so they are applied once (with the same Ctrl + b Space key strokes) but then may be freely resized and modified both horizontally and vertically.

Chapter 8

General Questions

sectionI forgot my root password, what do I do now?

The basic process to regain root is to boot into single user mode, mount the relevant partitions (/ and /usr), run `passwd(1)` to change the root password. You can then boot and login normally.

The detailed process:

- **Boot into single user mode**. This part of the process varies from platform to platform. For amd64 and i386 platforms, the second stage boot loader, boot(8), pauses for a few seconds to give you a chance to provide parameters to the kernel. This prompt looks like this:

  ```
  probing: pc0 com0 com1 apm mem[636k 190M a20=on]
  disk: fd0 hd0+
  >> OpenBSD/i386 BOOT 3.17
  boot>
  ```

  At this point, enter "`boot -s`" to bring the system up in single user mode:

  ```
  boot> boot -s
  ```

  Most other platforms send parameters to the kernel via the boot ROM. Of course the problem before this will probably be getting the system to shut down. Most likely, this will involve hitting the reset button or the power button. While hardly desirable, there usually isn't any alternative. Don't worry too much, OpenBSD's file system is very robust.

- **Mount the partitions**. Both "/" and /usr will need to be mounted read-write. Assuming they are on separate partitions (as they should be), the following will work:

  ```
  # fsck -p / && mount -uw /
  # fsck -p /usr && mount /usr
  ```

- **Run `passwd(1)` to change the root password**. As you already have root privileges (from being in single-user mode), it will not ask you to provide your current password.

- **boot into multiuser mode**. This can be done by either entering Ctrl + D to resume the normal boot process, or by entering the `reboot(8)` command.

If this is a non-personal machine, you should probably use `sudo(8)` to give multiple (trusted) people the ability to execute root commands.

"Wait. That looked too easy! That isn't very secure!" If an attacker has physical access to your system, they win, regardless of the OS on the computer. There are ways to force the use of a password on single-user mode (see

ttys(5)), or eliminate the pause on i386/amd64 (see boot.conf), but practically speaking, getting around those tricks is also pretty easy (One way: boot floppy or CDROM, edit or replace password file). You can try to prevent that, but then someone will pull the hard disk out of your computer. Making your computer difficult to manage properly isn't real security, and if you don't have the physical machine secured, you have no real security.

Note: many "remote management" systems give most of the functionality of physical access to the computer, and that needs to be considered. Don't tell yourself the system is secure if there is a way for an attacker to grab console, insert a virtual floppy and force a reboot of the machine. They might as well have physical access to the system. The console management system is likely not as secure as OpenBSD...

8.1 X won't start, I get lots of error messages

A common cause for X problems is the machdep.allowaperture sysctl(8) setting. Since this defaults to being disabled on OpenBSD, this is a fairly likely cause of the problem.

You need to edit */etc/sysctl.conf* and set machdep.allowaperture=2 (or 1, depending upon your platform). This will allow X to access the aperture driver, xf86(4), upon the next reboot. It can not be made available after boot. This can also be set during install if you answer "Y" when you are asked whether you expect to run the X Window System.

OpenBSD requires that the aperture driver be activated on alpha, amd64, i386, macppc and sparc64 platforms to control access to the video boards. Other platforms use a safer way to handle the video system, and do not need this (and do not have it in their kernel). If you do not anticipate using X on your system, it is recommended that you not enable the aperture driver.

For more information about configuring and using X on your platform, see the /usr/X11R6/README file on your installed system.

8.2 Can I use programming language "L" on OpenBSD?

You will find support for many common programming languages either in the base system (more specifically in the baseXX.tgz and compXX.tgz file sets), or in the packages and ports system. It is recommended that you install the required file set or package containing the specific compiler you want to use, instead of building it from source. For some compilers, building from source requires a lot of system resources and is often unneeded unless you have specific needs or there is no package available.

The following table attempts to give an overview of compilers for different languages, where you can find them, and whether there are any issues or limitations with them. Some of these are limited to certain platforms. This can be

seen either by examining a search result through the ports tree, and noting what is mentioned in "Archs", or by inspecting the port's Makefile directly. In the latter case, look for lines containing `ONLY_FOR_ARCHS`, `NOT_FOR_ARCHS`, `BROKEN`, etc.

Note: For ease of use, this article provides an alphabetical list, without distinguishing between different categories of programming languages. This is not a comprehensive list of everything that is available or can be used on OpenBSD. If you feel there are inaccuracies or issues which are not mentioned here, feel free to report that.

Language	Where?	Notes
Awk	base50.tgz, awk(1)	
	lang/gawk	GNU awk
C,C++	comp50.tgz, gcc(1)	The C/C++ compilers in the base system have been audited and they have several security enhancements (e.g. ProPolice) enabled by default. Please see gcc-local(1) for details. They will also emit warnings when using unsafe functions such as sprintf(), strcpy(), strcat(), tmpnam(), etc. Most platforms use gcc 4.2.1.
	lang/gcc, lang/llvm	These compilers have not gone through the security audit and do not contain security enhancements like those in the base system. The gcc binaries are renamed egcc, eg++, etc. to avoid confusion with their counterparts in the base system.
Caml	lang/ocaml	Objective Caml
COBOL	lang/open-cobol	
Erlan	lang/erlang	
Fortran	lang/g77	Only Fortran 77 support.
	lang/gcc	Fortran 95 is also supported by egfortran in gcc 4.0 and above. This new compiler is available as a subpackage (f95) of gcc.
Haskell	lang/ghc	
	lang/nhc98	
Java	devel/jdk	Sun JDK - only 1.7 as a package; for older version see build instructions below.
	lang/classpath	essential core class libraries for Java
	lang/kaffe	
	lang/jikes	Fast compiler, works well. This needs a "run-time jar", the bytecode version of all the standard API.
	devel/eclipse	Large IDE; works with Sun JDK
Lisp	lang/clisp	
Lua	lang/lua	Additional Lua libraries and auxiliary utilities are available in the ports tree.
Perl	base50.tgz,perl(1)	Many Perl modules are available in the ports tree, so search there first before installing modules from CPAN.
Perl 6	lang/rakudo	
PHP	www/php4	Plenty of subpackages are available for different PHP modules
	www/php5	
Prolog	lang/swi-prolog	SWI-Prolog environment.
Python	lang/python	Other ports are using Python 2.7 by default.
Ruby	lang/ruby	
Scheme	lang/chicken	
	lang/scheme48	
	lang/scm	
	shells/scsh	
Smalltalk	lang/squeak	
Tcl	lang/tcl	

Building the Sun JDK

Due to Sun's restrictive SCSL license, OpenBSD cannot ship binary packages for the JDK ¡ 1.7. Starting from 1.7 OpenBSD has a fully GPLv2 licensed port, that can be installed as a package. Users looking for the browser plugin will still need to build 1.5 or 1.6 from ports until Sun releases the plugin code. Note that you will need plenty of RAM for this build to succeed.

The JDK ports are in the `devel/jdk` subdirectory of the ports tree. You can choose among different versions, each in their own subdirectory. When you just type `make`, you will see a message asking you to to fetch the source files manually from Sun's website. Before you can do that, you need to register on that website, and agree with the license. That's why the ports framework cannot start the download automatically.

Once you have downloaded the necessary distribution files and patch sets, copy them to the `/usr/ports/distfiles` directory. You will also need to have X installed on your system. Start the build by issuing `make` in the port's subdirectory.

The JDK requires a working Java 2 compiler as a bootstrap to build. For this purpose, since OpenBSD 4.0, the port of JDK 1.5 uses kaffe, which allows JDK 1.5 to be used on both i386 and amd64 platforms, and reduces the build time considerably.

Older versions of the JDK still require a Linux version of the JDK. Linux emulation on OpenBSD is restricted to i386 systems, and so these older JDK versions will build only on i386. The ports framework should take care of installing the necessary files and setting `kern.emul.linux=1`. For more information, please read about Linux emulation in the `compat_linux(8)` manual page, and also FAQ 9 - Running Linux binaries on OpenBSD. Note that this Linux emulation is only required during the build of the JDK, which results in a native OpenBSD JDK. **You do not need Linux emulation to work with the native JDK**.

After many hours, the build will finish. Just continue with `make install` to install the JDK.

If you run into errors such as "Could not reserve enough space for object heap", try increasing your processes' memory limits using the shell's built-in `ulimit` command, with the -d flag.

Other development tools

Additionally, there are many other development tools available within the base system or as packages or ports. A few examples:

- Unix shells: `ksh` and `csh` in the base system, many others (e.g. `zsh`, `tcsh`) in the shells subdirectory of the ports tree. `lint(1)`: a C program verifier, which has been substantially improved from versions before OpenBSD 3.9. Linted versions of system libraries are also provided.

- "make" utilities: the traditional BSD make(1) program is in the base system, and the ports tree contains other flavors which are required to compile some software.
- Graphical toolkits: many popular graphical toolkits (e.g. GTK+, Tk, Qt, wxWidgets, ...) have been ported to OpenBSD. They can be found in the x11 subdirectory of the ports tree.
- Version control systems: GNU CVS as used by the OpenBSD project is in the base system, and the ports tree contains a few others. Watch for the new OpenCVS which is being developed.

8.3 What is the ports tree?

Please see chapter 15, Working with ports

8.4 What are packages?

Please see chapter 15, Package management.

8.5 Should I use Ports or Packages?

Please see chapter 15.

8.6 Is there any way to use my floppy drive if it's not attached during boot?

You need to set the kernel to always assume the floppy is attached, even if not detected during the hardware probe, by setting the 0x20 flag bit on fdc(4). This can be done by using User Kernel Config or config(8) to alter your kernel,

```
# config -e -f /bsd
OpenBSD 5.0 (GENERIC) #43: Wed Aug 17 10:10:52 MDT 2011
    deraadt@i386.openbsd.org:/usr/src/sys/arch/i386/compile/GENERIC
Enter 'help' for information
ukc> change fd*
254 fd* at fdc0 drive -1 flags 0x0
change [n] y
drive [-1] ? ENTER
flags [0] ? 0x20
254 fd* changed
254 fd* at fdc0 drive -1 flags 0x20
ukc> q
Saving modified kernel.
#
```

8.7 OpenBSD Bootloader (*i386, amd64 specific*)

When booting your OpenBSD system, you have probably noticed the boot prompt.

```
boot>
```

For most people, you won't have to do anything here. It will automatically boot if no commands are given. But sometimes problems arise, or special functions are needed. That's where these options will come in handy. To start off, you should read through the `boot(8)` man page. Here we will go over the most common used commands for the bootloader.

To start off, if no commands are issued, the bootloader will automatically try to boot `/bsd`. If that fails it will try `/obsd`, and if that fails, it will try `/bsd.old`. You can specify a kernel by hand by typing:

```
boot> boot hd0a:/bsd
```

or

```
boot> b /bsd
```

This will boot the kernel named bsd from the 'a' partition of the first BIOS recognized hard disk.

Here is a brief list of options you can use with the OpenBSD kernel.

- **-a** : This will allow you to specify an alternate root device after booting the kernel.
- **-c** : This allows you to enter the boot time configuration. Check the Boot Time Config section of the FAQ.
- **-s** : This is the option to boot into single user mode.
- **-d** : This option is used to dump the kernel into ddb. Keep in mind that you must have DDB built into the kernel.

These are entered in the format of: **boot** [**image** [**-acds**]]
For further reading you can read **boot(8)**'s man page.

8.8 S/Key

S/Key is a "one-time password" authentication system. It can be useful for people who don't have the ability to use an encrypted channel which protects their authentication credentials in transit, as can be established using `ssh(1)`.

WARNING: One-time password systems only protect authentication information. They do not prevent network eavesdroppers from gaining access to private information. Furthermore, if you are accessing a secure system A, it is

recommended that you do this from another trusted system B, to ensure nobody is gaining access to system A by logging your keystrokes or by capturing and/or forging input and output on your terminal devices.

The S/Key system generates a sequence of one-time (single use) passwords from a user's secret passphrase along with a challenge received from the server, by means of a secure hash function. The system is only secure if the secret passphrase is never transferred over the network. Therefore **initializing or changing your secret passphrase MUST be done over a secure channel**, such as `ssh(1)` or the console.

OpenBSD's S/Key implementation can use a variety of algorithms as the one-way hash function. The following algorithms are available:

- md4
- md5
- sha1
- rmd160

Setting up S/Key - The first steps

To start off the directory */etc/skey* must exist. If this directory is not in existence, have the super-user create it. This can be done simply by doing:

```
# skeyinit -E
```

Once that directory is in existence, you can initialize your S/Key. To do this you must use `skeyinit(1)`. Since `skeyinit(1)` will be asking you for your S/Key secret passphrase, you must run this over a secure channel, as explained above! The program will even remind you of this. With `skeyinit(1)`, you will first be prompted for your password to the system. This is the same password that you used to log into the system. Once you have authorized yourself with your system password, you will be asked for your S/Key secret passphrase. This is NOT your system password. Your secret passphrase must be at least 10 characters. We suggest using a memorable phrase containing several words as the secret passphrase. Here is an example user being added.

```
$ skeyinit
Reminder - Only use this method if you are directly connected
           or have an encrypted channel.  If you are using telnet,
           exit with no password and use skeyinit -s.
Password:
[Adding ericj with md5]
Enter new secret passphrase:
Again secret passphrase:

ID ericj skey is otp-md5 100 oshi45820
Next login password: HAUL BUS JAKE DING HOT HOG
```

One line of particular importance in here is ID *ericj skey is otp-md5 100 oshi45820*. This gives a lot of information to the user. Here is a breakdown of the sections and their importance.

- otp-md5 - This shows which one-way hash was used to create your One-Time Password (otp).
- 100 - This is your sequence number. This is a number from 100 down to 1. Once it reaches one, another secret passphrase must be created by running skeyinit(1).
- oshi45820 - This is the key.

But of more immediate importance is your one-time password. Your one-time password consists of 6 small words, combined together this is your one-time password, spaces and all. The one-time password printed by skeyinit cannot be used to login (there is a usage for this first one-time password, see `skeyinit(1)`). To be able to log in, a one-time password corresponding to the challenge printed by the login process has to be computed using `skey(1)`. The next section will show how to do that.

Actually using S/Key to login.

By now your skey has been initialized. You're ready to login. Here is an example session using S/Key to login. To perform an S/Key login, you append **:skey** to your login name.

```
$ ftp localhost
Connected to localhost.
220 oshibana.shin.ms FTP server (Version 6.5/OpenBSD) ready.
Name (localhost:ericj): ericj:skey
331- otp-md5 96 oshi45820
331 S/Key Password:
230- OpenBSD 5.0 (GENERIC) #43: Wed Aug 17 10:10:52 MDT 2011
230-
230- Welcome to OpenBSD: The proactively secure Unix-like operating system.
230-
230- Please use the sendbug(1) utility to report bugs in the system.
230- Before reporting a bug, please try to reproduce it with the latest
230- version of the code.  With bug reports, please try to ensure that
230- enough information to reproduce the problem is enclosed, and if a
230- known fix for it exists, include that as well.
230-
230 User ericj logged in.
Remote system type is UNIX.
Using binary mode to transfer files.
ftp> quit
221 Goodbye.
```

Note that I appended ":skey" to my username. This tells ftpd that I want to authenticate using S/Key. Some of you might have noticed that my sequence number has changed to *otp-md5 96 oshi45820*. This is because by now I have used S/Key to login several times. But how do you get your one-time password? Well, to compute the one-time password, you'll need to know what sequence number you're using and your key. As you're probably thinking, how can you remember which sequence number you're on?

When you are logging in, the login process prints a line containing the needed information, which you can use to generate a one-time password on the spot using another trusted computer accesses by a secure channel, by copy-pasting the line into a command shell:

```
otp-md5 96 oshi45820
```

After typing your passphrase, your one-time password will be printed, which you can then copy-paste to the S/Key Password prompt to log in. Not only is `otp-md5` a description of the hash used, it is also an alternate name for the `skey(1)` command.

If you already are logged in and want to generate a one-time password for the next login, use `skeyinfo(1)`, it will tell you what to use for the next login. For example here, I need to generate another one-time password for a login that I might have to make in the future. (remember I'm doing this from a secure channel).

```
$ skeyinfo
95 oshi45820
```

An even better way is to use `skeyinfo -v`, which outputs a command suitable to be run in the shell. For instance:

```
$ skeyinfo -v
otp-md5 95 oshi45820
```

So, the simplest way to generate the next S/Key password is just:

```
$ `skeyinfo -v`
Reminder - Do not use this program while logged in via telnet.
Enter secret passphrase:
NOOK CHUB HOYT SAC DOLE FUME
```

Note the backticks in the above example.

I'm sure many of you won't always have a secure connection or a trusted local computer to create these passwords, and creating them over an insecure connection isn't feasible, so how can you create multiple passwords at one time? You can supply **skey(1)** with a number of how many passwords you want created. This can then be printed out and taken with you wherever you go.

```
$ otp-md5 -n 5 95 oshi45820
Reminder - Do not use this program while logged in via telnet.
Enter secret passphrase:
91: SHIM SET LEST HANS SMUG BOOT
92: SUE ARTY YAW SEED KURD BAND
93: JOEY SOOT PHI KYLE CURT REEK
94: WIRE BOGY MESS JUDE RUNT ADD
95: NOOK CHUB HOYT SAC DOLE FUME
```

Notice here though, that the bottom password should be the first used, because we are counting down from 100.

Using S/Key with ssh(1) and telnet(1)

Using S/Key with `ssh(1)` or `telnet(1)` is done in pretty much the same fashion as with ftp–you simply tack ":skey" to the end of your username. Example:

```
$ ssh -l ericj:skey localhost
otp-md5 98 oshi45821
S/Key Password: SCAN OLGA BING PUB REEL COCA
Last login: Thu Apr  7 12:21:48 on ttyp1 from 156.63.248.77
OpenBSD 5.0 (GENERIC) #43: Wed Aug 17 10:10:52 MDT 2011

Welcome to OpenBSD: The proactively secure Unix-like operating system.

Please use the sendbug(1) utility to report bugs in the system.
Before reporting a bug, please try to reproduce it with the latest
version of the code.  With bug reports, please try to ensure that
enough information to reproduce the problem is enclosed, and if a
known fix for it exists, include that as well.

You have mail.
$
```

8.9 What OpenBSD platforms support SMP? (Symmetric Multi-Processor)

SMP is supported on the OpenBSD/i386, OpenBSD/amd64, OpenBSD/mvme88k, OpenBSD/sparc64 (including the UltraSPARC T1/T2/T2+ processors), OpenBSD/macppc, OpenBSD/sgi (Octane only), and OpenBSD/hppa platforms.

A separate SMP kernel, "`bsd.mp`", is provided with the install file sets. If multiple processors are detected at install time, is automatically installed as the default boot kernel, `/bsd`, and the single processor kernel is renamed `/bsd.sp`.

If you add additional processors to a system which was installed with only one processor, you will want to install the appropriate bsd.mp kernel, rename your existing `/bsd` to `/bsd.sp`, and rename `/bsd.mp` to `/bsd`.

It is hoped that other SMP-capable platforms will be supported in the future. On most other platforms, OpenBSD will run on an SMP system, but only utilizing one processor. The exception to this is the SPARC platform – OpenBSD/sparc will sometimes require that extra MBus modules be removed for the system to boot.

8.10 I get Input/output error when trying to use my tty devices

You need to use `/dev/cuaXX` for connections initiated from the OpenBSD system, the `/dev/ttyXX` devices are intended only for terminal or dial-in usage. While it was possible to use the tty devices in the past, the OpenBSD kernel is no longer compatible with this usage.

From `cua(4)`:

For hardware terminal ports, dial-out is supported through matching device nodes called calling units. For instance, the terminal called `/dev/tty03` would have a matching calling unit called `/dev/cua03`. These two devices are normally differentiated by creating the calling unit device node with a minor number 128 greater than the dial-in device node. *Whereas the dial-in device (the tty) normally requires a hardware signal to indicate to the system that it is active, the dial-out device (the cua) does not, and hence can communicate unimpeded with a device such as a modem.* This means that a process like `getty(8)` will wait on a dial-in device until a connection is established. Meanwhile, a dial-out connection can be established on the dial-out device (for the very same hardware terminal port) without disturbing anything else on the system. The `getty(8)` process does not even notice that anything is happening on the terminal port. If a connecting call comes in after the dial-out connection has finished, the `getty(8)` process will deal with it properly, without having noticed the intervening dial-out action.

8.11 What web browsers are available for OpenBSD?

Lynx, a text-based browser, is in the base system, and has SSL support. Other browsers available include (in no particular order):

Graphical (X) Browsers

- Konqueror Installed as part of the KDE desktop environment.
- Links+ Another fast and small graphical browser. (Also has a text-only mode)
- Firefox and SeaMonkey Feature-filled browsers. SeaMonkey includes many non-browser features (mail client, IRC client, etc.), Firefox is just a browser, based on Mozilla.
- Opera Commercial browser, i386 only (requires Linux emulation).

- Amaya The W3C's browser and editor.
- Midori, a WebKit-based browser from the Xfce project.
- Chromium the open source version of the Google Chrome browser, i386/amd64 only (so far).
- XXXTerm a minimalist browser with vi-like keyboard operations in addition to traditional browser behavior
- Conkeror a keyboard-oriented, highly-customizable, highly-extensible web browser based on Mozilla.
- Surf Extremely minimal webbrowser based on WebKit/GTK+.
- Dillo Fast and lightweight, based on Fast Light ToolKit and custom rendering engine (note: no javascript support).
- Netsurf Lightweight web browser based on custom rendering engine (note: no javascript support).
- Epiphany GNOME web browser based on WebKit.
- Arora Simple Qt4-based browser using WebKit.

Console (Text mode) Browsers

- Elinks Feature-rich, can render both frames and tables, highly customizable.
- W3M Has table and frame support (also has a graphical mode).
- Links Has table support.
- Retawk Interactive, multi-threaded text mode web browser.

Most of the abovementioned browsers are available as pre-compiled packages on the FTP servers, with some also on the CD-ROM. After the installation of the ports tree, all can be found it the `/usr/ports/www/` directory.

As most of the graphical browsers are very large and require quite some time to download and compile, one should *seriously* consider the use of packages where available.

8.12 How do I use the mg editor?

`mg(1)` is a micro Emacs-style text editor included in OpenBSD. Micro means that it's a small implementation which is mostly similar to the text editor features of Emacs 17. It does not implement many of Emacs' other features (including mail and news functionality, as well as modes for Lisp, C++, Lex, Awk, Java, etc...).

A concise tutorial is included with OpenBSD (located at `/usr/src/usr.bin/mg/tutorial`).

8.13 ksh(1) does not appear to read my .profile!

There are two likely reasons for ksh(1) to seemingly ignore a user's .profile file.

- .profile is not owned by the user. To fix for username,

  ```
  # chown username ~username/.profile
  ```

- You are using ksh(1) from within X Window System

  Under xterm(1), argv[0] for ksh(1) is not prepended with a dash ("-"). Prepending a dash to argv[0] will cause csh(1) and ksh(1) to know they should interpret their login files. (For csh(1) that's .login, with a separate .cshrc that is always run when csh(1) starts up. With ksh(1), this is more noticeable because there is only one startup script, .profile. This file is ignored unless the shell is a login shell.)

  To fix this, add the line "XTerm*loginShell: true" to the file .Xdefaults in your home directory. Note, this file does not exist by default, you may have to create it.

  ```
  $ echo "XTerm*loginShell: true" >> ~/.Xdefaults
  ```

  You may not have had to do this on other systems, as some installations of X Window System come with this setting as default. OpenBSD has chosen to follow the X.org behavior.

8.14 Why does my /etc/motd file get overwritten when I modified it?

The /etc/motd file is edited upon every boot of the system, replacing everything up to, but not including, the first blank line with the system's kernel version information. When editing this file, make sure that you start after this blank line, to keep /etc/rc from deleting these lines when it edits /etc/motd upon boot.

8.15 Antialiased and TrueType fonts in X

see Anti-aliasing and TrueType Fonts on OpenBSD document.

8.16 Does OpenBSD support any journaling filesystems?

No it doesn't. We use a different mechanism to achieve similar results called Soft Updates. Please read chapter 14 - Soft Updates to get more details.

8.17 Reverse DNS or Why is it taking so long for me to log in?

Many new users to OpenBSD experience a two minute login delay when using services such as `ssh` or `ftp`. This can also be experienced when using a proxy, such as `ftp-proxy`, or when sending mail out from a workstation through `sendmail`.

This is almost always due to a reverse-DNS problem. DNS is Domain Name Services, the system the Internet uses to convert a name, such as "www.openbsd.org" into a numeric IP address. Another task of DNS is the ability to take a numeric address and convert it back to a "name", this is "Reverse DNS".

In order to provide better logging, OpenBSD performs a reverse-DNS lookup on any machine that attaches to it in many different ways, including `ssh`, `ftp`, `sendmail`, or `ftp-proxy`. Unfortunately, in some cases, the machine that is making the connection does not have a proper reverse DNS entry.

An example of this situation:

A user sets up an OpenBSD box as a firewall and gateway to their internal home network, mapping all their internal computers to one external IP using NAT. They may also use it as an outbound mail relay. They follow the installation guidelines, and are very happy with the results, except for one thing – every time they try to attach to the box in any way, they end up with a two minute delay before things happen.

What is going on:

From a workstation behind the NAT of the gateway with an unregistered IP address of 192.168.1.35, the user uses `ssh` to access the gateway system. The `ssh` client prompts for username and password, and sends them to the gateway box. The gateway then tries to figure out who is trying to log in by performing a reverse DNS lookup of 192.168.1.35. The problem is 192.168.0.0 addresses are for private use, so a properly configured DNS server outside your network knows it should have no information about those addresses. Some will quickly return an error message, in these cases, OpenBSD will assume there is no more information to be gained, and it will quickly give up and just admit the user. Other DNS servers will not return ANY response. In this case you will find yourself waiting for the OpenBSD name resolver to time out, which takes about two minutes before the login will be permitted to continue. In the case of `ftp-proxy`, some ftp clients will timeout before the reverse DNS query times out, leading to the impression that ftp-proxy isn't working.

This can be quite annoying. Fortunately, it is an easy thing to fix.

Fix, using /etc/hosts:

The simplest fix is to populate your /etc/hosts file with all the workstations you have in your internal network, and ensure that your /etc/resolv.conf file contains the line lookup file bind which ensures that the resolver knows to start with the /etc/hosts file, and failing that, to use the DNS servers specified by the "nameserver" lines in your /etc/resolv.conf file.

Your /etc/hosts file will look something like this:

```
::1 localhost.in.example.org localhost
127.0.0.1 localhost.in.example.org localhost
192.168.1.1 gw.in.example.org gw
192.168.1.20 scrappy.in.example.org scrappy
192.168.1.35 shadow.in.example.org shadow
```

Your resolv.conf file will look something like this:

```
search in.example.org
nameserver 24.2.68.33
nameserver 24.2.68.34
lookup file bind
```

A common objection to this is "But, I use DHCP for my internal network! How can I configure my /etc/hosts?" Rather easily, actually. Just enter lines for all the addresses your DHCP server is going to give out, plus any static devices:

```
::1 localhost.in.example.org localhost
127.0.0.1 localhost.in.example.org localhost
192.168.1.1 gw.in.example.org gw
192.168.1.20 scrappy.in.example.org scrappy
192.168.1.35 shadow.in.example.org shadow
192.168.1.100 d100.in.example.org d100
192.168.1.101 d101.in.example.org d101
192.168.1.102 d102.in.example.org d102
         [... snip ...]
192.168.1.198 d198.in.example.org d198
192.168.1.199 d199.in.example.org d199
```

In this case, I am assuming you have the DHCP range set to 192.168.1.100 through 192.168.1.199, plus the three static definitions as listed at the top of the file.

If your gateway must use DHCP for configuration, you may well find you have a problem – dhclient will overwrite your /etc/resolv.conf every time the lease is renewed, which will remove the "lookup file bind" line. This can be solved by putting the line "lookup file bind" in the file /etc/resolv.conf.tail.

Fix, using a local DNS server

Details on this are somewhat beyond the scope of this document, but the basic trick is to setup your favorite DNS server, and make sure it knows it is authoritative for both forward and reverse DNS resolution for all nodes in your network, and make sure your computers (including your gateway) know to use it as a DNS server.

8.18 Why do the OpenBSD web pages not conform to HTML4/XHTML?

The present web pages have been carefully crafted to work on a wide variety of actual browsers going back to browser versions 4.0 and later. We do not want to make these older pages conform to HTML4 or XHTML until we're sure that they will also work with older browsers; it's just not a priority. We welcome new contributors, but suggest you work on writing code, or on documenting new aspects of the system, not on tweaking the existing web pages to conform to newer standards.

8.19 Why is my clock off by twenty-some seconds?

When using `rdate(8)` to synchronize your clock to a NTP server, you may find your clock is off by twenty-some seconds from your local definition of time.

This is caused by a difference between the UTC (Coordinated Universal Time, based on astronomical observations) time and TAI (International Atomic Time, based on atomic clocks) time. To compensate for variations in the earth's rotation, "leap seconds" are inserted into UTC, but TAI is unadjusted. These leap seconds are

Addressing the problem is fairly simple. In most countries you will get the correct time if you use the "`-c`" parameter to `rdate(8)` and use a time zone out of the directory `/usr/share/zoneinfo/right/`. For example, if you are located in Germany, you could use these commands:

```
# cd /etc && ln -sf /usr/share/zoneinfo/right/CET localtime
    # rdate -ncv ptbtime1.ptb.de
```

In other countries, the rules may differ.

8.20 Why is my clock off by several hours?

By default, OpenBSD assumes your hardware clock is set to UTC (Universal Coordinated Time) rather than local time, assumed by some other operating systems, which can cause problems when multi-booting.

Many other operating systems, can be configured to do the same, which avoids this problem altogether.

If having the hardware clock set to UTC is a problem, you can change the default behavior of OpenBSD using `config(8)`. For example, to configure OpenBSD to use a hardware clock set to US/Eastern (5 hours behind UTC, so 300 minutes):

```
# config -ef /bsd
OpenBSD 5.0 (GENERIC) #43: Wed Aug 17 10:10:52 MDT 2011
    deraadt@i386.openbsd.org:/usr/src/sys/arch/i386/compile/GENERIC
Enter 'help' for information
ukc> timezone 300
timezone = 300, dst = 0
ukc> quit
Saving modified kernel.
```

See `options(4)` and search for option "TIMEZONE=value" for more information.

Normally, the time zone is set during install. If you have need to change the time zone, you can create a new symbolic link to the appropriate time zone file in `/usr/share/zoneinfo`. For example, to set the machine to use `EST5EDT` as the new local time zone:

```
# ln -fs /usr/share/zoneinfo/EST5EDT /etc/localtime
```

See also:

- `date(1)`
- "Why is my clock off by twenty-some seconds?"
- OpenBSD's NTPD

Chapter 9

Migrating to OpenBSD

9.1 Tips for users of other Unix-like Operating Systems

While OpenBSD is a very traditional Unix-like operating system and will be very familiar to those who have used other Unix-like systems, there are important differences. New users to OpenBSD must look at their own experience: if your only knowledge of Unix is some experience with one variant of Linux, you may find OpenBSD "strange". Rest assured, Linux looks pretty strange to anyone who starts from OpenBSD. You must recognize the difference between "standard" and your experience.

If you learned Unix from any of the good books on general Unix, understanding the "Unix philosophy" and then extended your knowledge to a particular platform, you will find OpenBSD to be a very "true" and familiar Unix. If you learned Unix using a "type this to do that" process or a book such as "Learn PinkBeenie v8.3 in 31.4 Hours", and told yourself you "know Unix", you will most likely find OpenBSD very different.

One important difference between OpenBSD and many other operating systems is the documentation. OpenBSD developers take great pride in the system man pages. The man pages are the authoritative source of OpenBSD documentation – not this FAQ, not third-party independently maintained pages, not "HOWTO"s, etc. When a developer makes a change to the system, they are expected to update the man pages along with their change to the system code, not "later" or "when they get around to it" or "when someone complains". A manual page exists for virtually every program, utility, driver, configuration file, and so on on the stock system. It is expected that a user will check the man pages before asking for help on the mail lists.

Here are some of the commonly encountered differences between OpenBSD and other Unix variants.

- OpenBSD is a fairly pure "BSD-Style" Unix, following the 4.4BSD design closely. Linux and SCO Unix are "System V" style systems. Some Unix-like operating systems (including some Linux distributions) mix many SysV and BSD characteristics. A common place where this causes confusion is the startup scripts, OpenBSD uses the traditional BSD4.4-style `rc(8)` style.

- OpenBSD is a complete *system*, intended to be kept in sync. It is not a "Kernel plus utilities" that can be upgraded separately from each other. Failure to keep your system (kernel, user utilities, and applications) in sync will result in bad things happening.

- As many applications are not developed to directly compile and run on an OpenBSD environment, OpenBSD has a ports tree, a system where users can easily acquire code, patch it for OpenBSD, install dependencies, compile it, install and remove it in a standardized and maintainable way. Pre-compiled packages are created and distributed by the OpenBSD ports

team. Users are encouraged to use these packages over compiling their own.

- OpenBSD uses CVS to keep track of source code changes. OpenBSD pioneered anonymous CVS, which allows anyone to extract the full source tree for any version of OpenBSD (from 2.0 to current, and all revisions of all files in between) at any time, and you can access the most recent changes within hours of its commit. There is also a very convenient and easy to use web interface to CVS.

- OpenBSD produces an official release available on CD and FTP every six months on a predefined schedule. Snapshots for all supported platforms are made semi-regularly with the current development code. It is the goal that the source tree is kept fully buildable and the resultant system usable at all times. The tree is occasionally broken, but this is an extraordinary event that will be corrected rapidly, not something that will be permitted to continue.

- OpenBSD contains strong cryptography, which can not be included with OSs based in some countries.

- OpenBSD has gone through heavy and continual security auditing to ensure the quality (and thus, security) of the code.

- OpenBSD's kernel is /bsd.

- The names of hard disks are usually `/dev/wd` (IDE) and `/dev/sd` (SCSI or devices emulating SCSI disks).

- `/sbin/route` with no arguments in Linux gives the state of all the active routes, under OpenBSD (and many other OSs), you need the "`show`" parameter, or do a "`netstat -r`".

- OpenBSD does NOT support Journaling Filesystems like ReiserFS, IBM's JFS or SGI's XFS. Instead we use the Soft Updates feature of the already very robust Unix Fast File System (FFS) to accomplish the goals of performance and stability.

- OpenBSD comes with Packet Filter (PF), not ipfw, ipchains, netfilter, iptables, or ipf. This means that Network Address Translation (known as IP-Masquerading in Linux), queuing, and filtering is done through `pfctl(8)`, `pf(4)`, and `pf.conf(5)`. See the PF User's Guide for detailed configuration information.

- Interface address is stored in /etc/hostname.¡interfacename¿ (for example, `/etc/hostname.dc0` for a NIC using the `dc(4)` driver). It can contain a hostname (resolved in `/etc/hosts`) instead of an IP address.

- The machine name is in `/etc/myname`.

- The default gateway is in `/etc/mygate`.

- OpenBSD's default shell is `/bin/ksh`, which is pdksh, the Public Domain Korn shell. Other included shells are `csh` and `sh`. Shells such as *bash* and *tcsh* can be added as packages or installed from ports. Users familiar with *bash* are encouraged to try `ksh(1)` before loading *bash* on their system – it does what most people desire of *bash*.

- Password management on OpenBSD is different from password management on some other Unix-like operating systems. The actual passwords are stored in the file `master.passwd(5)` which is readable only by root. This file should be altered only with the `vipw` program.

- Devices are named by driver, not by type. For example, there are no eth* devices. It would be ne0 for an NE2000 Ethernet card, and `xl0` for a 3Com Etherlink XL or a Fast Etherlink XL Ethernet device, etc. All of these drivers have man pages in section 4. So, to find more information about the messages your 3c905 driver is putting out, you can do "`man 4 xl`".

- OpenBSD/i386, amd64, and several other platforms use a "two layer" disk partitioning system, where the first layer is the `fdisk`, BIOS-visible partition, familiar to most users of IBM compatible computers. The second layer is the `disklabel`, a traditional BSD partitioning system. OpenBSD supports up to 15 disklabel partitions on a disk, all residing within one fdisk partition. This permits OpenBSD to coexist with other OSs, including other Unix-like OSs. OpenBSD must be one of the four "primary" partitions.

- Some other OSs encourage you to customize your kernel for your machine. OpenBSD users are encouraged to simply use the standard GENERIC kernel provided and tested by the developers. Users attempting to "customize" or "optimize" their kernel usually cause far more problems than they solve, and will not be supported by developers.

- OpenBSD works hard to maintain the license policy and security of the project. For this reason, some newer versions of some software which fail to meet either the license or security goals of the project have not and may never be integrated into OpenBSD. Security and free licensing will never take a back seat to having the biggest version number.

9.2 Dual booting Linux and OpenBSD

Yes! It is possible! Read INSTALL.linux.

9.3 Converting your Linux (or other Sixth Edition-style) password file to BSD-style

First, figure out if your Linux password file is shadowed or not. If it is, install John the Ripper from packages or ports (`security/john`) and use the unshadow utility that comes with it to merge your `passwd` and `shadow` files into one Sixth Edition-style file.

Using your Linux password file, we'll call it `linux_passwd`, you need to add in ::0:0 between fields four and seven. `awk(1)` does this for you.

```
# cat linux_passwd | awk -F : '{printf("%s:%s:%s:%s::0:0:%s:%s:%s\n", \
> $1,$2,$3,$4,$5,$6,$7); }' > new_passwd
```

At this point, you want to edit the `new_passwd` file and remove the root and other system entries that are already present in your OpenBSD password file or aren't applicable with OpenBSD (all of them). Also, make sure there are no duplicate usernames or user IDs between `new_passwd` and your OpenBSD box's `/etc/passwd`. The easiest way to do this is to start with a fresh `/etc/passwd`.

```
# cat new_passwd >> /etc/master.passwd
# pwd_mkdb -p /etc/master.passwd
```

The last step, `pwd_mkdb` is necessary to rebuild the `/etc/spwd.db` and `/etc/pwd.db` files. It also creates a Sixth Edition-style password file (minus encrypted passwords) at `/etc/passwd` for programs which use it. OpenBSD uses a stronger encryption for passwords, blowfish, which is very unlikely to be found on any system which uses full Sixth Edition-style password files. To switch over to this stronger encryption, simply have the users run ' `passwd`' and change their password. The new password they enter will be encrypted with your default setting (usually blowfish unless you've edited `/etc/login.conf`). Or, as *root*, you can run `passwd` *username*.

9.4 Running Linux binaries on OpenBSD

OpenBSD/i386 is able to run Linux binaries when the kernel is compiled with the `COMPAT_LINUX` option and the runtime sysctl `kern.emul.linux` is also set. If you are using the `GENERIC` kernel (which you should be), `COMPAT_LINUX` is already enabled, and you will just need to do:

```
# sysctl kern.emul.linux=1
```

For this to be done automatically each time the computer boots, remove the # (comment) character at the beginning of the line

```
#kern.emul.linux=1      # enable running Linux binaries
```

in `/etc/sysctl.conf`, so that it reads

```
kern.emul.linux=1       # enable running Linux binaries
```

and reboot your system to have it take effect.

To run any Linux binaries that are not statically linked (most of them), you need to follow the instructions on the `compat_linux(8)` manual page.

A simple way to get most of the useful Linux libraries is to install the `fedora/base` package from your nearest FTP mirror. To find out more about the packages and ports system read FAQ 15 - The OpenBSD Packages and Ports System. To install the above mentioned package you would issue

```
# export PKG_PATH=ftp://your.ftp.mirror/pub/OpenBSD/5.0/packages/i386/
# pkg_add -i fedora_base
```

Note that `pkg_add(1)` will automatically execute sysctl to set `kern.emul.linux` to the correct value upon adding this package. However, it does not change /etc/sysctl.conf, so if you wish Linux emulation to be enabled by default, you need to modify `kern.emul.linux` there.

9.5 Accessing your Linux files from OpenBSD

OpenBSD supports the EXT2FS file system. For further information, see chapter 14.

Chapter 10

System Management

10.1 When I try to su to root it says that I'm in the wrong group

On OpenBSD, users who are in the `wheel` group are allowed to use `su(1)` to become root. Otherwise, the user will receive an error.

If you are creating new users with `adduser(8)`, you can add them to the `wheel` group by answering "`wheel`" at the "`Invite user into other groups:`" prompt. Existing users must be added to the `wheel` group by hand. Here is an example of a `/etc/group` entry which has had the user **ericj** added to the `wheel` group:

```
wheel:*:0:root,ericj
```

If you want to give access to superuser privileges without adding users to the `wheel` group, use `sudo(8)`.

10.2 How do I duplicate a filesystem?

To duplicate your filesystem use `dump(8)` and `restore(8)`. For example, to duplicate everything under directory `SRC` to directory `DST`, do a:

```
# cd /SRC; dump 0f - . | (cd /DST; restore -rf - )
```

dump is designed to give you plenty of backup capabilities, and it may be an overkill if you just want to duplicate a part of a (or an entire) filesystem. The command `tar(1)` may be faster for this operation. The format looks very similar:

```
# cd /SRC; tar cf -  . | (cd /DST; tar xpf - )
```

10.3 How do I start daemons with the system? (Overview of `rc(8)`)

OpenBSD itself uses an `rc(8)` style startup. This uses a few key files for startup.

- `/etc/rc` - Main script. Should not be edited.
- `/etc/rc.conf` - Configuration file used by */etc/rc* to set startup parameters for the system. Should not be edited.
- `/etc/rc.conf.local` - Configuration file that overrides settings in `/etc/rc.conf` so you don't have to touch `/etc/rc.conf` itself, which is important when upgrading your system.
- `/etc/netstart` - Script used to initialize the network. Shouldn't be edited.

- `/etc/rc.local` - Script used for local administration. This is where new daemons or host specific information can be set.
- `/etc/rc.securelevel` - Script which runs commands that must be run before the security level changes. See `init(8)`
- `/etc/rc.shutdown` - Script run on shutdown. Put anything you want done before shutdown in this file. See `rc.shutdown(8)`

How does `rc(8)` work?

The main files a system administrator should concentrate on are */etc/rc.conf* (for guidance), */etc/rc.conf.local* (for changes), */etc/rc.local* and */etc/rc.shutdown*. To get a look of how the `rc(8)` procedure works, here is the flow:

After the kernel is booted, */etc/rc* is started:

- Filesystems are checked.
- Default configuration variables are read in from */etc/rc.conf*, then local changes to those variables are read from */etc/rc.conf.local*. Settings in rc.conf.local will override those in rc.conf.
- Filesystems are mounted
- Clears out */tmp* and preserves any editor files
- Configures the network via */etc/netstart*

  Configures your interfaces up.

  Sets your hostname, domainname, etc.
- Starts system daemons
- Performs various other checks (quotas, savecore, etc)
- Runs */etc/rc.firsttime*
- Runs */etc/rc.local*
- Processes local and package scripts in */etc/rc.d*

Starting Daemons and Services that come with OpenBSD

Most daemons and services that come with OpenBSD are controlled on boot by variables defined in the */etc/rc.conf* configuration file. Take a look at the `/etc/rc.conf` file. You'll see lines similar to this:

```
ftpd_flags=NO           # for non-inetd use: ftpd_flags="-D"
```

A line like this shows that `ftpd(8)` is not to start up with the system (at least not as a daemon via `rc(8)`; ftpd is often run out of `inetd(8)`, see Anonymous FTP FAQ to read more about this). Each line has a comment showing you the flags for common usage of that daemon or service. This doesn't mean that you must run that daemon or service with those flags. Read the relevant manual page to see how you can have that daemon or service start up in any way you like.

We strongly suggest you do not alter */etc/rc.conf* itself. Instead, create or edit the file */etc/rc.conf.local*, copy just the lines you need to change from */etc/rc.conf* and adjust them as you like. This makes future upgrades easier – all the changes are in the one file that isn't touched during upgrade. In fact, the standard upgrade process assumes you have not modified */etc/rc.conf*, and will overwrite it with the new version.

For example, here is the default line pertaining to `httpd(8)`.

```
httpd_flags=NO          # for normal use: "" (or "-DSSL" after reading ssl(8))
```

Here you can see that starting up httpd normally requires no flags, so a line like *httpd_flags=""* added to */etc/rc.conf.local* will do the job. But to start httpd with SSL enabled (refer to the SSL FAQ or `ssl(8)`), you would start with a line like `httpd_flags="-DSSL"`, though you may need to add other parameters for other reasons.

Starting up local daemons and configuration

For other daemons which you might install on the system via packages or other ways, you could use the */etc/rc.local* file. For example, I've installed a daemon which lies at `/usr/local/sbin/daemonx`. I want it to start at boot time. I could put an entry into */etc/rc.local* like this:

```
if [ -x /usr/local/sbin/daemonx ]; then
echo 'Starting daemonx'; /usr/local/sbin/daemonx
fi
```

(If the daemon does not automatically detach on startup, remember to add a "`&`" at the end of the command line.)

From now on, this daemon will be started at boot. You will be able to see any errors on boot, a normal boot with no errors would show a line like this:

```
Starting daemonx
```

The */etc/rc.d/* directory

Starting with OpenBSD 5.0, system daemons ("services") are started, stopped and controlled by */etc/rc.d*. All system daemons are handled by these scripts, and most add-in packages are as well.

These scripts, one per daemon, are invoked by `rc`. The order for system daemons is coded into rc, and the order for add-in packages is managed by the `pkg_scripts` environment variable, which would be set in `/etc/rc.conf.local`. Note that simply placing a script in this directory does not cause it to be run on boot; the name of the script must be specified in the `pkg_scripts` variable to start on boot.

The starting of system scripts is determined by entries in the `/etc/rc.conf.local` file. For example, `/etc/rc.d/httpd` does not start `httpd(1)` unless `/etc/rc.conf` or `/etc/rc.conf.local` contains a line defining the "`httpd_flags`" variable. To help make sure your system will come up as expected on the next boot, the `rc.d` scripts will not run their daemon if the appropriate variable is not defined. You can, of course, manually invoke `/usr/sbin/httpd` directly with whatever options you wish, if you wish to run the program manually.

Note that rather than having each script in rc.d managing the entire startup, shutdown, reload, restart, and check operations, most `rc.d` scripts can be reduced to specifying a very few variables, and invoking the `rc.subr` script, which manages most of the standard way of doing these tasks.

For example, our above `daemonx` application could be started with a `/etc/rc.d` file consisting of:

```
#!/bin/sh

daemon="/usr/local/sbin/daemonx"

. /etc/rc.d/rc.subr

rc_cmd $1
```

and adding the daemon name to the `pkg_scripts` variable in `/etc/rc.conf.local`.

rc.shutdown

/etc/rc.shutdown is a script that is run at shutdown. Anything you want done before the system shuts down should be added to this file. If you have apm, you can also set "`powerdown=YES`", which will give you the equivalent of "`shutdown -p`".

10.4 Why do users get "relaying denied" when they are remotely sending mail through my OpenBSD system?

Try this:

```
# grep relay-domains /etc/mail/sendmail.cf
```

The output may look something like this:

```
FR-o /etc/mail/relay-domains
```

If this file doesn't exist, create it. You will need to enter the hosts who are sending mail remotely with the following syntax:

```
.domain.com    #Allow relaying for/to any host in domain.com
sub.domain.com #Allow relaying for/to sub.domain.com and any host in that domain
10.2           #Allow relaying from all hosts in the IP net 10.2.*.*
```

Don't forget send a 'HangUP' signal to sendmail (a signal which causes most daemons to re-read their configuration file):

```
# kill -HUP `head -1 /var/run/sendmail.pid`
```

Further Reading

- http://www.sendmail.org/~ca/email/relayingdenied.html
- http://www.sendmail.org/tips/relaying.php
- http://www.sendmail.org/antispam/

10.5 I've set up POP, but users have trouble accessing mail through POP. What can I do?

Most issues dealing with POP are problems with temporary files and lock files. If your pop server sends an error message such as:

```
-ERR Couldn't open temporary file, do you own it?
```

Try setting up your permissions as such:

```
permission in  /var
drwxrwxr-x   2 bin     mail     512 May 26 20:08 mail


permissions in  /var/mail
-rw-------   1 username   username        0 May 26 20:08 username
```

Another thing to check is that the user actually owns their own /var/mail file. Of course this should be the case (as in, /var/mail/joe should be owned by joe) but if it isn't set correctly it could be the problem!

Of course, making /var/mail writable by group mail opens up some vague and obscure security problems. It is likely that you will never have problems

with it. But it could (especially if you are a high profile site, ISP, ...)! There are several POP servers you can install right away from the ports collection. If possible, use `popa3d` which is available in the OpenBSD base install. Or, you could just have the wrong options selected for your pop daemon (like dot locking). Or, you may just need to change the directory that it locks in (although then the locking would only be valuable for the POP daemon.)

Note: OpenBSD does not have a group name of "mail". You need to create this in your /etc/group file if you need it. An entry like:

```
mail:*:6:
```

would be sufficient.

10.6 Why does Sendmail ignore the /etc/hosts file?

By default, Sendmail uses DNS for name resolution, not the `/etc/hosts` file. The behavior can be changed through the use of the `/etc/mail/service.switch` file.

If you wish to query the hosts file before DNS servers, create a `/etc/mail/service.switch` file which contains the following line:

```
hosts       files dns
```

If you wish to query ONLY the hosts file, use the following:

```
hosts       files
```

Send Sendmail a HUP signal:

```
# kill -HUP `head -1 /var/run/sendmail.pid`
```

and the changes will take effect.

10.7 Setting up a Secure HTTP server with SSL(8)

OpenBSD ships with an SSL-ready httpd and RSA libraries. For use with `httpd(8)`, you must first have a certificate created. This will be kept in */etc/ssl/* with the corresponding key in */etc/ssl/private/*. The steps shown here are taken in part from the `ssl(8)` man page. Refer to it for further information. This FAQ entry only outlines how to create an RSA certificate for web servers, not a DSA server certificate. To find out how to do so, please refer to the `ssl(8)` man page.

To start off, you need to create your server key and certificate using OpenSSL:

```
# openssl genrsa -out /etc/ssl/private/server.key 1024
```

Or, if you wish the key to be encrypted with a passphrase that you will have to type in when starting servers

```
# openssl genrsa -des3 -out /etc/ssl/private/server.key 1024
```

The next step is to generate a Certificate Signing Request which is used to get a Certifying Authority (CA) to sign your certificate. To do this use the command:

```
# openssl req -new -key /etc/ssl/private/server.key -out /etc/ssl/private/server.csr
```

This *server.csr* file can then be given to Certifying Authority who will sign the key. One such CA is **Thawte Certification** which you can reach at http://www.thawte.com/.

If you cannot afford this, or just want to sign the certificate yourself, you can use the following.

```
# openssl x509 -req -days 365 -in /etc/ssl/private/server.csr  -signkey /etc/ssl/priva
```

With */etc/ssl/server.crt* and */etc/ssl/private/server.key* in place, you should be able to start `httpd(8)` with the **-DSSL** flag (see the section about `rc(8)` in this faq), enabling https transactions with your machine on port 443.

10.8 I edited /etc/passwd, but the changes didn't seem to take place. Why?

If you edit */etc/passwd* directly, your changes will be lost. OpenBSD generates */etc/passwd* dynamically with `pwd_mkdb(8)`. The main password file in OpenBSD is */etc/master.passwd*. According to `pwd_mkdb(8)`,

```
FILES
     /etc/master.passwd  current password file
     /etc/passwd         a 6th Edition-style password file
     /etc/pwd.db         insecure password database file
     /etc/pwd.db.tmp     temporary file
     /etc/spwd.db        secure password database file
     /etc/spwd.db.tmp    temporary file
```

In a traditional Unix password file, such as /etc/passwd, everything including the user's encrypted password is available to anyone on the system (and is a prime target for programs such as Crack). 4.4BSD introduced the master.passwd file, which has an extended format (with additional options beyond those provided by /etc/passwd) and is only readable by root. For faster access to data, the library calls which access this data normally read /etc/pwd.db and /etc/spwd.db.

OpenBSD does come with a tool with which you should edit your password file. It is called `vipw(8)`. Vipw will use vi (or your favourite editor defined per

$EDITOR) to edit /etc/master.passwd. After you are done editing, it will recreate /etc/passwd, /etc/pwd.db, and /etc/spwd.db as per your changes. Vipw also takes care of locking these files, so that if anyone else attempts to change them at the same time, they will be denied access.

10.9 What is the best way to add and delete users?

OpenBSD provides two commands for easily adding users to the system:

- `adduser(8)`
- `user(8)`

You can also add users by hand, using `vipw(8)`, but this is more difficult for most operations.

The easiest way to add a user in OpenBSD is to use the `adduser(8)` script. You can configure `adduser(8)` by editing */etc/adduser.conf*. `adduser(8)` allows for consistency checks on */etc/passwd*, */etc/group*, and shell databases. It will create the entries and $HOME directories for you. It can even send a message to the user welcoming them. Here is an example user, **testuser**, being added to a system. He/she will be given the $HOME directory /home/testuser, made a member of the group **guest**, and given the shell */bin/ksh*.

```
# adduser
Use option ‘‘-silent’’ if you don’t want to see all warnings and questions.

Reading /etc/shells
Check /etc/master.passwd
Check /etc/group

Ok, let’s go.
Don’t worry about mistakes. There will be a chance later to correct any input.
Enter username []: testuser
Enter full name []: Test FAQ User
Enter shell csh ksh nologin sh [ksh]: ksh
Uid [1002]: Enter
Login group testuser [testuser]: guest
Login group is ‘‘guest’’. Invite testuser into other groups: guest no
[no]: no
Login class authpf daemon default staff [default]: Enter
Enter password []: Type password, then Enter
Enter password again []: Type password, then Enter

Name:        testuser
Password:    ****
```

```
Fullname:    Test FAQ User
Uid:         1002
Gid:         31 (guest)
Groups:      guest
Login Class: default
HOME:        /home/testuser
Shell:       /bin/ksh
OK? (y/n) [y]: y
Added user ``testuser''
Copy files from /etc/skel to /home/testuser
Add another user? (y/n) [y]: n
Goodbye!
```

To delete users you should use the `rmuser(8)` utility. This will remove all existence of a user. It will remove any `crontab(1)` entries, their $HOME dir (if it is owned by the user), and their mail. Of course it will also remove their */etc/passwd* and */etc/group* entries. Next is an example of removing the user that was added above. Notice you are prompted for the name, and whether or not to remove the user's home directory.

```
# rmuser
Enter login name for user to remove: testuser
Matching password entry:

testuser:$2a$07$ZWnBOsbqMJ.ducQBfsTKUe3PL97Ve1AHWJ0A4uLamniLNXLeYrEie:1002
:31::0:0:Test FAQ User:/home/testuser:/bin/ksh

Is this the entry you wish to remove? y
Remove user's home directory (/home/testuser)? y
Updating password file, updating databases, done.
Updating group file: done.
Removing user's home directory (/home/testuser): done.
```

Adding users via `user(8)`

These tools are less interactive than the `adduser(8)` command, which makes them easier to use in scripts.

The full set of tools is:

- `group(8)`
- `groupadd(8)`
- `groupdel(8)`
- `groupinfo(8)`
- `groupmod(8)`

- user(8)
- useradd(8)
- userdel(8)
- userinfo(8)
- usermod(8)

Actually adding users

The command */usr/sbin/user* is just a frontend to the rest of the */usr/sbin/user** commands. Therefore, the following commands can be added by using **user add** or **useradd**, which form you chose doesn't change the results at all. Remember, since user(8) is not interactive, the easiest way to add users is with adduser(8).

useradd(8) is less daunting to use if you know the default settings beforehand. These settings are located in `usermgmt.conf(5)` and can be viewed by doing:

```
$ user add -D
group           users
base_dir        /home
skel_dir        /etc/skel
shell           /bin/csh
inactive        0
expire          Null (unset)
range           1000..60000
```

These defaults will be used unless you specify alternatives with the command line options. For example, we want the user to be added to the group **guest**, not **users**. One more little hurdle with adding users, is that passwords must be specified on the command line. Importantly, the passwords must be encrypted, so you need to use the `encrypt(1)` utility. For example: OpenBSD's passwords by default use the Blowfish algorithm for 6 rounds. Here is an example to create an encrypted password to give to useradd(8).

```
$ encrypt -p -b 6
Enter string:
$2a$06$YOdOZM3.4m6MObBXjeZtBOWArqC2.uRJZXUkOghbieIvSWXVJRzlq
```

Now that we have an encrypted password, we are ready to add the user. We will add the same user with the same specifications as the user we added above, via adduser(8).

```
# user add -p '$2a$06$YOdOZM3.4m6MObBXjeZtBOWArqC2.uRJZXUkOghbieIvSWXVJRzlq' -u 1002  -s /bin/ksh
```

Note: Make sure to use ' ' (single quotes) around the password string, not " " (double quotes) as the shell will interpret these before sending it to user(8).

In addition to that, make sure you specify the **-m** option if you want the user's home directory created and the files from */etc/skel* copied over.

To see that the user was created correctly, we can use many different utilities. Below are a few commands you can use to quickly check that everything was created correctly.

```
$ ls -la /home
total 14
drwxr-xr-x   5 root      wheel   512 May 12 14:29 .
drwxr-xr-x  15 root      wheel   512 Apr 25 20:52 ..
drwxr-xr-x  24 ericj     wheel  2560 May 12 13:38 ericj
drwxr-xr-x   2 testuser  guest   512 May 12 14:28 testuser
$ id testuser
uid=1002(testuser) gid=31(guest) groups=31(guest)
$ finger testuser
Login: testuser                         Name: Test FAQ User
Directory: /home/testuser               Shell: /bin/ksh
Last login Sat Apr 22 16:05 (EDT) on ttyC2
No Mail.
No Plan.
```

In addition to these commands, user(8) provides its own utility to show user characteristics, called userinfo(8).

```
$ userinfo testuser
login   testuser
passwd  *
uid     1002
groups  guest
change  Wed Dec 31 19:00:00 1969
class
gecos   Test FAQ User
dir     /home/testuser
shell   /bin/ksh
expire  Wed Dec 31 19:00:00 1969
```

Removing users

To remove users with the user(8) hierarchy of commands, you will use userdel(8). This is a very simple, yet usable command. To remove the user created in the last example, simply:

```
# userdel -r testuser
```

Notice the **-r** option, which must be specified if you want the users home directory to be deleted as well. Alternatively, you can specify **-p** and not **-r** and this will lock the user's account, but not remove any information.

10.10 How do I create an ftp-only account (not anonymous FTP!)?

There are a few ways to do this, but a very common way to do such is to add "`/usr/bin/false`" into "`/etc/shells`". Then when you set a users shell to "`/usr/bin/false`", they will not be able log in interactively, but will be able to use ftp capabilities. You may also want to restrict access by Confining users to their home directory in ftpd.

10.11 Setting up Quotas

Quotas are used to limit user's space that they have available to them on your disk drives. It can be very helpful in situations where you have limited resources. Quotas can be set by user and/or by group.

The first step to setting up quotas is to make sure that "`option QUOTA`" is in your Kernel Configuration. This option is in the GENERIC kernel. After this, you need to mark in `/etc/fstab` the filesystems which will have quotas enabled. The keywords `userquota` and `groupquota` should be used to mark each filesystem that you will be using quotas on. By default, the files `quota.user` and `quota.group` will be created at the root of that filesystem to hold the quota information. This default can be overridden by specifying the file name with the quota option in `/etc/fstab`, such as "`userquota=/var/quotas/quota.user`". Here is an example `/etc/fstab` that has one filesystem with userquotas enabled, and the quota file in a non-standard location:

```
/dev/wd0a / ffs rw,userquota=/var/quotas/quota.user 1 1
```

Now it's time to set the user's quotas. To do so you use the utility `edquota(8)`. A simple use is just "`edquota <user>`". `edquota(8)` will use **`vi(1)`** to edit the quotas unless the environmental variable EDITOR is set to a different editor. For example:

```
# edquota ericj
```

This will give you output similar to this:

```
Quotas for user ericj:
/: KBytes in use: 62, limits (soft = 0, hard = 0)
        inodes in use: 25, limits (soft = 0, hard = 0)
```

To add limits, edit it to give results like this:

```
Quotas for user ericj:
/: KBytes in use: 62, limits (soft = 1000, hard = 1050)
        inodes in use: 25, limits (soft = 0, hard = 0)
```

Note that the quota allocation is in 1k blocks. In this case, the softlimit is set to 1000k, and the hardlimit is set to 1050k. A softlimit is a limit where the user is just warned when they cross it and have until their grace period is up to get their disk usage below their limit. Grace periods can be set by using the **-t** option on edquota(8). After the grace period is over the softlimit is handled as a hardlimit. This usually results in an allocation failure.

Now that the quotas are set, you need to turn the quotas on. To do this use `quotaon(8)`. For example:

```
# quotaon -a
```

This will go through `/etc/fstab` to turn on the filesystems with quota options. Now that quotas are up and running, you can view them using `quota(1)`. Using a command of "`quota <user>`" will give that user's information. When called with no arguments, the quota(1) command will give your quota statistics. For example:

```
# quota ericj
```

Will result in output similar to this:

```
Disk quotas for user ericj (uid 1001):
     Filesystem  blocks   quota   limit   grace   files   quota   limit   grace
               /      62    1000    1050              27       0       0
```

By default quotas set in `/etc/fstab` will be started on boot. To turn them off use

```
# quotaoff -a
```

10.12 Setting up KerberosV Clients and Servers

OpenBSD includes KerberosV as a pre-installed component of the default system.

For more information on KerberosV, from your OpenBSD system, use the command:

```
# info heimdal
```

10.13 Setting up Anonymous FTP Services

Anonymous FTP allows users without accounts to access files on your computer via the File Transfer Protocol. This will give an overview of setting up the anonymous FTP server, and its logging, etc.

Adding the FTP account

To start off, you need to have an *ftp* account on your system. This account should not have a usable password. Here we will set the login directory to /home/ftp, but you can put it wherever you want. When using anonymous ftp, the ftp daemon will chroot itself to the home directory of the *ftp* user. To read up more on that, read the `ftpd(8)` and `chroot(2)` man pages. Here is an example of adding the ftp user. I will do this using `adduser(8)`. We also need to add /usr/bin/false to our */etc/shells*, this is the "shell" that we will be giving to the *ftp* user. This won't allow them to login, even though we will give them an empty password. To do this you can simply do

```
echo /usr/bin/false >> /etc/shells
```

After this, you are ready to add the *ftp* user:

```
# adduser
Use option ‘‘-silent’’ if you don’t want to see all warnings and questions.

Reading /etc/shells
Check /etc/master.passwd
Check /etc/group

Ok, let’s go.
Don’t worry about mistakes. There will be a chance later to correct any input.
Enter username []: ftp
Enter full name []: anonymous ftp
Enter shell csh false ksh nologin sh [ksh]: false
Uid [1002]: Enter
Login group ftp [ftp]: Enter
Login group is ‘‘ftp’’. Invite ftp into other groups: guest no
[no]: no
Login class authpf daemon default staff [default]: Enter
Enter password []: Enter
Set the password so that user cannot logon? (y/n) [n]: y

Name:        ftp
Password:    ****
Fullname:    anonymous ftp
Uid:         1002
Gid:         1002 (ftp)
Groups:      ftp
Login Class: default
HOME:        /home/ftp
Shell:       /usr/bin/false
OK? (y/n) [y]: y
Added user ‘‘ftp’’
```

```
Copy files from /etc/skel to /home/ftp
Add another user? (y/n) [y]: n
Goodbye!
```

Directory Setup

Along with the user, this created the directory `/home/ftp`. This is what we want, but there are some changes that we will have to make to get it ready for anonymous ftp. Again these changes are explained in the `ftpd(8)` man page.

You **do not** need to make a /home/ftp/usr or /home/ftp/bin directory.

- */home/ftp* - This is the main directory. It should be owned by root and have permissions of 555.
- */home/ftp/etc* - This is entirely optional and not recommended, as it only serves to give out information on users which exist on your box. If you want your anonymous ftp directory to appear to have real users attached to your files, you should copy /etc/pwd.db and /etc/group to this directory. This directory should be mode 511, and the two files should be mode 444. These are used to give owner names as opposed to numbers. There are no passwords stored in pwd.db, they are all in spwd.db, so don't copy that over.
- */home/ftp/pub* - This is a standard directory to place files in which you wish to share. This directory should also be mode 555.

Note that all these directories should be owned by "root". Here is a listing of what the directories should look like after their creation.

```
# pwd
/home
# ls -laR ftp
total 5
dr-xr-xr-x  5 root  ftp    512 Jul  6 11:33 .
drwxr-xr-x  7 root  wheel  512 Jul  6 10:58 ..
dr-x--x--x  2 root  ftp    512 Jul  6 11:34 etc
dr-xr-xr-x  2 root  ftp    512 Jul  6 11:33 pub

ftp/etc:
total 43
dr-x--x--x  2 root  ftp    512 Jul  6 11:34 .
dr-xr-xr-x  5 root  ftp    512 Jul  6 11:33 ..
-r--r--r--  1 root  ftp    316 Jul  6 11:34 group
-r--r--r--  1 root  ftp  40960 Jul  6 11:34 pwd.db

ftp/pub:
total 2
```

```
dr-xr-xr-x  2 root  ftp  512 Jul  6 11:33 .
dr-xr-xr-x  5 root  ftp  512 Jul  6 11:33 ..
```

Starting up the server and logging

You can choose to start ftpd either by `inetd(8)` or from the rc scripts. These examples will show our daemon being started from inetd.conf. First we must become familiar with some of the options to ftpd. The default line from */etc/inetd.conf* is:

```
ftp             stream  tcp     nowait  root    /usr/libexec/ftpd       ftpd -US
```

Here ftpd is invoked with *-US*. This will log anonymous connections to */var/log/ftpd* and concurrent sessions to */var/run/utmp*. That will allow for these sessions to be seen via who(1). For some, you might want to run only an anonymous server, and disallow ftp for users. To do so you should invoke ftpd with the *-A* option. Here is a line that starts ftpd up for anonymous connections only. It also uses *-ll* which logs each connection to syslog, along with the get, retrieve, etc., ftp commands.

```
ftp             stream  tcp     nowait  root    /usr/libexec/ftpd       ftpd -llUSA
```

Note: For people using HIGH traffic ftp servers, you might not want to invoke ftpd from inetd.conf. The best option is to comment the ftpd line from inetd.conf and start ftpd from rc.conf.local along with the *-D* option. This will start ftpd as a daemon, and has much less overhead as starting it from inetd. Here is an example line to start it from rc.conf.local.

```
ftpd_flags="-DllUSA"            # for non-inetd use: ftpd_flags="-D"
```

This of course only works if you have ftpd taken out of */etc/inetd.conf* and made inetd re-read its configuration file.

It is not necessary to give additional startup options to ftpd to allow anonymous connections. The preceding steps of creating the 'ftp' user and configuring the relevant directories with the correct permissions are enough. However, to stop anonymous connections it isn't necessary to undo everything. Just restart ftpd with the **-n** option included. Anonymous connections are then disabled.

Other relevant files

/etc/ftpwelcome - This holds the Welcome message for people once they have connected to your ftp

/etc/motd - This holds the message for people once they have successfully logged into your ftp s

.message - This file can be placed in any directory. It will be shown once a user enters that di

10.14 Confining users to their home directories in `ftpd(8)`

By default, when logging in by ftp, users can change to any directory on the filesystem that they have access to. This may not be desirable in some cases. It is possible to restrict what users may see through ftp sessions by chrooting them to their home directory.

If you only wish to allow chrooted ftp logins, use the **-A** option to `ftpd(8)`.

If you wish to apply them more finely, OpenBSD's login capability infrastructure and `ftpd(8)` together make this easy.

Users in a login class with the `ftp-chroot` variable set are automatically chrooted. Additionally, you can add a username to the file `/etc/ftpchroot` to chroot those usernames. A user only needs to be listed in one of these locations.

10.15 Applying patches in OpenBSD

Even with OpenBSD, bugs happen. Some bugs may lead to reliability issues (i.e., something may cause the system to stop functioning as desired). Other bugs may lead to security vulnerabilities (which may allow others to "use" your computer in unintended and unauthorized ways). When a critical bug is found, the fix will be committed to the *-current* source tree, and patches will be released for the supported releases of OpenBSD. These patches appear on the errata web page, and are separated into "common" errata that impact all platforms, and errata that impact only one or more, but not all, platforms.

Note, however, that patches aren't released for new features or additional hardware support for OpenBSD, and are only published for important reliability fixes or security problems that should be addressed right away on impacted systems (which is often NOT all systems, depending on their purpose).

There are three ways to update your system with patched code:

- **Upgrade your system to *-current***. As all fixes are applied to the -current code base, updating your system to the latest snapshot is a very good way to apply fixed code. However, running -current is not for everyone.
- **Update your system to *-stable***. This is done by fetching or updating your source tree using the appropriate -stable branch, and recompiling the kernel and userland files. Overall, this is probably the easiest way, though it takes longer (as the entire system gets recompiled) and a complete source checkout can take a long time if you have limited bandwidth available.
- **Patch, compile and install individual impacted files**. This is what we will use for our example below. While this requires less bandwidth and typically less time than an entire cvs(1) checkout/update and source code compilation, this is sometimes the most difficult option, as there is no one universal set of instructions to follow. Sometimes you must

patch, recompile and install one application, other times, you might have to recompile entire sections of the tree if the problem is in a library file.

Again, patching individual files is not always simple, so give serious thought to following the *-stable* (or "patch") branch of OpenBSD. Mixing and matching of patching solutions can be done if you understand how everything works, but new users should pick one method and stick with it.

How are "errata" patches different from what is in the CVS tree?

All patches posted to the errata web page are patches directly against the indicated release's source tree. Patches against the latest CVS tree might also include other changes that wouldn't be wanted on a release system. This is important: If you have installed a snapshot, checked out the source trees at the time you obtained that snapshot and attempt to patch it using a published patch, you may well find the patch doesn't apply, as that code may have changed.

Applying patches.

Patches for the OpenBSD Operating System are distributed as "Unified diffs", which are text files that hold differences to the original source code. They are **NOT** distributed in binary form. This means that to apply a patch you must have the source code from the **RELEASE** version of your system available. In general, it is advisable to aquire the entire source tree before applying a patch. If you are running a release from an official CDROM, the source trees are available as files on disk 3, these can also be downloaded from the FTP servers. We will assume you have the entire tree checked out.

For our example here, we will look at patch 001 for OpenBSD 3.6 dealing with the `st(4)` driver, which handles tape drives. Without this patch, recovering data from backups was quite difficult. People using a tape drive *needed* this patch, however those without a tape drive may have had no particular need to install it. Let's look at the patch:

```
# more 001_st.patch
Apply by doing:
        cd /usr/src
        patch -p0 < 001_st.patch

Rebuild your kernel.

Index: sys/scsi/st.c
===================================================================
RCS file: /cvs/src/sys/scsi/st.c,v
retrieving revision 1.41
```

```
retrieving revision 1.41.2.1
diff -u -p -r1.41 -r1.41.2.1
--- sys/scsi/st.c       1 Aug 2004 23:01:06 -0000       1.41
+++ sys/scsi/st.c       2 Nov 2004 01:05:50 -0000       1.41.2.1
@@ -1815,7 +1815,7 @@ st_interpret_sense(xs)
        u_int8_t skey = sense->flags & SSD_KEY;
        int32_t info;

-       if (((sense->flags & SDEV_OPEN) == 0) ||
+       if (((sc_link->flags & SDEV_OPEN) == 0) ||
            (serr != 0x70 && serr != 0x71))
                return (EJUSTRETURN); /* let the generic code handle it */
```

As you will note, the top of the patch includes brief instructions on applying it. We will assume you have put this patch into the /usr/src directory, in which case, the following steps are used:

```
# cd /usr/src
# patch -p0 < 001_st.patch
Hmm...  Looks like a unified diff to me...
The text leading up to this was:
--------------------------
|Apply by doing:
|        cd /usr/src
|        patch -p0 < 001_st.patch
|
|Rebuild your kernel.
|
|Index: sys/scsi/st.c
|===================================================================
|RCS file: /cvs/src/sys/scsi/st.c,v
|retrieving revision 1.41
|retrieving revision 1.41.2.1
|diff -u -p -r1.41 -r1.41.2.1
|--- sys/scsi/st.c      1 Aug 2004 23:01:06 -0000       1.41
|+++ sys/scsi/st.c      2 Nov 2004 01:05:50 -0000       1.41.2.1
--------------------------
Patching file sys/scsi/st.c using Plan A...
Hunk #1 succeeded at 1815.              <-- Look for this message!
done
```

Note the "Hunk #1 succeeded" message above. This indicates the patch was applied successfully. Many patches are more complex than this one, and will involve multiple hunks and multiple files, in which case, you should verify that all hunks succeeded on all files. If they did not, it normally means your source tree was different in some way from the release source tree the patch was created from, or you didn't follow instructions carefully, or your patch was

mangled. Patches are very sensitive to "white space" – copying and pasting from your browser will often change tab characters into spaces or otherwise alter the white space of a file, making it not apply.

At this point, you can build and install the new kernel, then reboot the system as normal.

Not all patches are for the kernel. In some cases, you will have to rebuild individual utilities. At other times, will require recompiling all utilities statically linked to a patched library. Follow the guidance in the header of the patch, and if uncertain, rebuild the entire system.

Patches that are irrelevant to your particular system need not be applied – usually. For example, if you did not have a tape drive on your system, you would not benefit from the above patch. However, patches are assumed to be applied "in order" – it is possible that a later patch is dependent upon an earlier one. Be aware of this if you elect to "pick and choose" which patches you apply, and if in doubt, apply them all, in order.

10.16 Tell me about this `chroot(2)` Apache?

In OpenBSD, the Apache `httpd(8)` server has been `chroot(2)`ed by default. While this is a tremendous boost to security, it can create issues, if you are not prepared.

What is a chroot?

A `chroot(2)`ed application is locked into a particular directory and unable to wander around the rest of the directory tree, and sees that directory as its "/" (root) directory. In the case of `httpd(8)`, the program starts, opens its log files, binds to its TCP ports (though, it doesn't accept data yet), and reads its configuration. Next, it locks itself into */var/www* and drops privileges, then starts to accept requests. This means all files served and used by Apache must be in the */var/www* directory. In the default configuration of OpenBSD, all the files in the */var/www* directory are read-only by the user Apache runs as, *www*. This helps security tremendously – should there be a security issue with Apache, the damage will be confined to a single directory with only "read only" permissions and no resources to cause mischief with.

What does this mean to the administrator?

Put bluntly, chroot(2)ing Apache is something not done by default in most other operating systems. Many applications and system configurations will not work in a chroot(2) without some customization. Further, it must be remembered that security and convenience are often not compatible goals. OpenBSD's implementation of Apache does not compromise security for features or "ease".

- **Historic file system layouts**: Servers upgraded from older versions of OpenBSD may have web files located in user's directories, which clearly

won't work in a `chroot(2)`ed environment, as **httpd(8)** can't reach the */home* directory. Administrators may also discover their existing */var/www* partition is too small to hold all web files. Your options are to restructure or do not use the chroot(2) feature. You can, of course, use symbolic links in the user's home directories pointing to subdirectories in */var/www*, but you can NOT use links in */var/www* pointing to other parts of the file system – that is prevented from working by the chroot(2)ing. Note that if you want your users to have chroot(2)ed FTP access, this will not work, as the FTP chroot will (again) prevent you from accessing the targets of the symbolic links. A solution to this is to not use /home as your home directories for these users, rather use something similar to */var/www/users*. Symbolic links can be used completely within the chroot(2), but they have to be relative, not absolute.

- **Log Rotation**: Normally, logs are rotated by renaming the old files, then sending httpd(8) a SIGUSR1 signal to cause Apache to close its old log files and open new ones. This is no longer possible, as httpd(8) has no ability to open log files for writing once privileges are dropped. httpd(8) must be stopped and restarted. It sometimes takes a few seconds for all the child processes to terminate, which must happen before httpd(8) can be restarted, so one possible way to rotate the logs would be as follows:

```
# apachectl stop
    rename your log files
# apachectl start ; sleep 10 ; apachectl start
```

  Yes, the last line attempts to restart Apache immediately, and in case that fails it waits a few seconds and tries again. And yes, that does mean that for a few seconds every time you do your log rotation, your web server will be unavailable. While this could be annoying, any attempt to permit httpd(8) to reopen files after chroot(2)ing would defeat the very purpose of the chroot! There are also other strategies available, including logging to a pipe(2), and using an external log rotator at the other end of the pipe(2).

- **Existing Apache modules**: Virtually all will load, however some may not work properly in chroot(2), and many have issues on "`apachectl restart`", generating an error, which causes httpd(8) to exit.

- **Existing CGIs**: Most will NOT work as is. They may need programs or libraries outside */var/www*. Some can be fixed by compiling so they are statically linked (not needing libraries in other directories), most may be fixed by populating the */var/www* directory with the files required by the application, though this is non-trivial and requires some knowledge of the program.

- **File system mount options**: By default in OpenBSD, your */var* partition will be mounted with the `nosuid` and `nodev` options. If you attempt

to use an application within the chroot, you may need to change those options. You may need to do that even if you don't use the chroot option, of course.

- **Name Resolution**: httpd(8) inside the chroot(2) will NOT be able to use the system */etc/hosts* or */etc/resolv.conf*. Therefore, if you have applications which require name resolution, you will need to populate */var/www/etc/hosts* and/or */var/www/etc/resolv.conf* in the chroot(2) environment. Note that some applications expect the resolution of "localhost" to work.

- **Timezone**: httpd(8) inside the chroot(2) will NOT be able to use the system */etc/localtime*. If you require localtime logging of events, you will need to copy (not link) the corresponding timezone from */usr/share/zoneinfo/* under */var/www/etc/localtime*.

In some cases, the application or configuration can be altered to run within the chroot(2). In other cases, you will simply have to disable this feature using the -u option for httpd(8) in */etc/rc.conf.local*.

Example of chroot(2)ing an app: wwwcount

As an example of a process that can be used to chroot an application, we will look at wwwcount, a simple web page counter available through packages. While a very effective program, it knows nothing about chroot(2)ed Apache, and will not work chroot(2)ed in its default configuration.

First, we install the wwwcount package. We configure it and test it, and we find it doesn't seem to work, we get an Apache message saying "Internal Server Error". First step is to stop and restart Apache with the -u switch to verify that the problem is the chroot(2)ing, and not the system configuration.

```
# apachectl stop
/usr/sbin/apachectl stop: httpd stopped
# httpd -u
```

After doing this, we see the counter works properly, at least after we change the ownership on a directory so that Apache (and the CGIs it runs) can write to the files it keeps. So, we definitely have a chroot problem, so we stop and restart Apache again, using the default chrooting:

```
# apachectl stop
/usr/sbin/apachectl stop: httpd stopped
# httpd
```

A good starting point would be to assume wwwcount uses some libraries and other files it can't get to in the chroot. We can use the ldd(1) command to find out the dynamic object dependencies that the CGI needs:

```
# cd /var/www/cgi-bin/
# ldd Count.cgi
Count.cgi:
        Start    End      Type Open Ref GrpRef Name
        1c000000 3c007000 exe  1    0   0      /var/www/cgi-bin/Count.cgi
        0c085000 2c0be000 rlib 0    1   0      /usr/lib/libc.so.57.0
        08713000 08713000 rtld 0    1   0      /usr/libexec/ld.so
```

Ok, here is a problem, two files that are not available in the **chroot(2)** environment. So, we copy them over:

```
# mkdir -p /var/www/usr/lib /var/www/usr/libexec
# cp /usr/lib/libc.so.57.0 /var/www/usr/lib
# cp /usr/libexec/ld.so /var/www/usr/libexec
```

and try the counter again.

Well, now the program is running at least, and giving us error messages directly: "Unable to open config file for reading". Progress, but not done yet. The configuration file is normally in `/var/www/wwwcount/conf`, but within the chroot environment, that would seem to be `/wwwcount/conf`. Our options are to either recompile the program to make it work where the files are now, or move the data files. As we installed from a package, we'll just move the data file. In order to use the same config either chroot(2)ed or not, we'll use a symbolic link:

```
# mkdir -p /var/www/var/www
# cd /var/www/var/www
# ln -s ../../wwwcount wwwcount
```

Note that the symbolic link is crafted to work within the chroot. Again, we test... and we find we have yet another issue. Now wwwcount is complaining that it can't find the "strip image" files it uses to display messages. After a bit of searching, we find those are stored in /usr/local/lib/wwwcount, so we have to copy those into the chroot, as well.

```
# tar cf - /usr/local/lib/wwwcount | (cd /var/www; tar xpf - )
```

we test again... and it works!

Note that we have copied over only files that are absolutely required for operation. In general, only the minimum files needed to run an application should be copied into the chroot.

Should I use the chroot feature?

In the above example, the program is fairly simple, and yet we have seen several different kinds of problems.

Not every application can or should be chroot(2)ed.

The goal is a secure web server, chroot(2)ing is just a tool to accomplish this, it is not the goal itself. Remember, the starting configuration of the OpenBSD

chroot(2)ed Apache is where the user the httpd(8) program is running as can not run any programs, can not alter any files, and can not assume another user's identity. Loosen these restrictions, you have lessened your security, chroot or no chroot.

Some applications are pretty simple, and chroot(2)ing them makes sense. Others are very complex, and are either not worth the effort of forcing them into a chroot(2), or by the time you copy enough of the system into the chroot, you have lost the benefit of the chroot(2) environment. For example, the OpenWebMail program requires the ability to read and write to the mail directory, the user's home directory, and must be able to work as any user on the system. Attempting to push it into a chroot would be completely pointless, as you would end up disabling all the benefits of chroot(2)ing. Even with an application as simple as the above counter, it must write to disk (to keep track of its counters), so some benefit of the chroot(2) is lost.

Any application which has to assume root privileges to operate is pointless to attempt to chroot(2), as root can generally escape a chroot(2).

Do not forget, if the chrooting process for your application is too difficult, you may not upgrade or update the system as often as you should. This could end up making your system LESS secure than a more maintainable system with the chroot feature deactivated.

10.17 Can I change the root shell?

It is sometimes said that one should never change the root shell, though there is no reason not to in OpenBSD.

The default shell for *root* on OpenBSD is `ksh`.

A traditional Unix guideline is to only use statically compiled shells for root, because if your system comes up in single user mode, non-root partitions won't be mounted and dynamically linked shells won't be able to access libraries located in the `/usr` partition. This isn't actually a significant issue for OpenBSD, as the system will prompt you for a shell when it comes up in single user mode, and the default is `sh`. The three standard shells in OpenBSD (`csh`, `sh` and `ksh`) are all statically linked, and thus usable in single user mode.

10.18 What else can I do with *ksh*?

In OpenBSD, `ksh` is pdksh, the Public Domain Korn Shell, and is the same binary as `sh`.

Users comfortable with *bash*, often used on Linux systems, will probably find `ksh` very familiar. `Ksh(1)` provides most of the commonly used features in bash, including tab completion, command line editing and history via the arrow keys, and CTRL-A/CTRL-E to jump to beginning/end of the command line. If other features of *bash* are desired, *bash* itself can be loaded via either packages or ports.

The command prompt of ksh can easily be changed to something providing more information than the default "`$ `" by setting the `PS1` variable. For example, inserting the following line:

```
export PS1='$PWD $ '
```

in your `/etc/profile` produces the following command prompt:

```
/home/nick $
```

See the file `/etc/ksh.kshrc`, which includes many useful features and examples, and may be invoked in your user's `.profile`.

OpenBSD's ksh(1) has been enhanced with a number of "special characters" for the primary prompt string, PS1, similar to those used in bash. For example:

\e Insert an ASCII escape character.

\h The hostname, minus domain name.

\H The full hostname, including domain name.

\n Insert a newline character.

\t The current time, in 24-hour HH:MM:SS format.

\u The current user's username.

\w The current working directory. $HOME is abbreviated as ' '.

\W The basename of the current working directory.

\$ Displays "#" for root users, "$" for non-root users.

(see the **ksh(1)** man page for more details, and many, many more special characters! Also note the "$" character has special meaning inside double quotes, so handle it carefully)

One could use the following command:

```
export PS1="\n\u@\H\n\w \\$ "
```

to give an overly verbose but somewhat useful prompt.

10.19 Directory services

OpenBSD can be used for both servers and clients of databases containing user credentials, group information and other network-related data.

10.19.1 Which directory services are available?

Of course, you could use various directory services on OpenBSD. But YP is the only one that can be accessed directly using standard C-library functions like `getpwent(3)`, `getgrent(3)`, `gethostbyname(3)` and so on. Thus, if you keep your data in a YP database, you do not need to copy it to local configuration files like `master.passwd(5)` before you can use it, for example to authenticate system users.

YP is a directory service compatible with Sun Microsystems NIS (Network Information System). See `yp(8)` for an overview of the available manual pages. Be careful, some operating systems contain directory services bearing similar names but all the same being incompatible, for example NIS+.

To use other directory services except YP, you either need to populate local configuration files from the directory, or you need a YP frontend to the directory. For example, you can use the `sysutils/login_ldap` port when you choose the former, while the `ypldap(8)` daemon provides the latter.

For some applications, simply synchronizing a small number of configuration files among a group of machines using tools like `cron(8)`, `scp(1)` or `rsync` (available from ports) constitutes an easy and robust alternative to a full-blown directory service.

10.19.2 YP security considerations

For compatibility reasons, all security features built into the OpenBSD implementation of YP are switched *off* by default. Even when they are all switched on, the NIS protocol is still inherently insecure for two reasons: All data, including sensitive data like password hashes, is transmitted unencrypted across the network, and neither the client nor the server can reliably verify each other's identity.

Thus, before setting up any YP server, you should consider whether these inherent security flaws are acceptable in your context. In particular, YP is inadequate if potential attackers have physical access to your network. Anybody gaining root access to any computer connected to your network segments carrying YP traffic can bind your YP domain and retrieve its data. In some cases, passing YP traffic through SSL or IPSec tunnels might be an option, or you might consider combining YP with `kerberos(8)` authentication.

10.19.3 Setting up a YP server

1. A YP server serves a group of clients called a "domain". You should first select a domain name; it can be an arbitrary string and need not be related in any way to DNS domain names. Choosing a random name like "Eepoo5vi" can marginally improve security, though the effect is mostly in security by obscurity. In case you need to maintain several distinct YP domains, it's probably better to choose descriptive names like "sales", "marketing" and "research" in order to forestall system administration

errors caused by obscurity. Also note that some versions of SunOS require using the host's DNS domain name, so your choice might be restricted in a network including such hosts.

Use the `domainname(1)` utility to set the domain name, and put it into the file `defaultdomain(5)` to have it automatically set at system startup time.

```
echo "puffynet" > /etc/defaultdomain
domainname ‘cat /etc/defaultdomain‘
```

2. Initialise the YP server using the interactive command

   ```
   ypiniy -m
   ```

   At this point, it is not necessary to specify slave servers yet. To add slave servers, you can rerun `ypinit(8)` later, using the `-u` option. Setting up at least one slave server for each domain is useful to avoid service interruptions, should the master server ever go down or lose network connectivity, in particular since client processes trying to access YP maps block indefinitely until they receive the requested information. Thus, YP service interruptions typically render the client hosts completely unusable until YP is back to service.

3. Decide where to store the source files to generate your YP maps from. Keeping the server configuration separate from the served configuration helps to control which information will be served and which won't, so the default `/etc` often isn't the best choice.

   The only inconvenience caused by changing the source directory is that you will not be able to add, remove and modify users and groups in the YP domain using utilities like `user(8)` and `group(8)`. Instead, you will have to edit the configuration files with a text editor.

   To define the source directory, edit the file `/var/yp/‘domainname‘/Makefile` and change the `DIR` variable, e.g.

   ```
   DIR=/etc/yp/src/puffynet
   ```

4. Consider customizing other variables in `/var/yp/‘domainname‘/Makefile`. See `Makefile.yp(8)` for details.

   For example, even in case you use the default source directory `/etc`, you do not usually need all accounts and groups existing on the server on all your client hosts. In particular, not serving the root account and thus keeping root's password hash confidential is often beneficial to security. Review the values of `MINUID`, `MAXUID`, `MINGID` and `MAXGID` and adjust them to you needs.

If all your YP clients run OpenBSD or FreeBSD, exclude the encrypted passwords from the `passwd` maps by setting `UNSECURE=""` in **/var/yp/‘domainname‘/Makefile**.

The former practice of editing the template file `/var/yp/Makefile.yp` is no longer recommended. Changes to that file affect all domains initialized after the change, but do not affect domains initialized before the change, so this is error-prone either way: You both risk that the intended changes do not take effect, and you risk to forget about them and have them affect other domains later which they were never intended for.

5. Create the source directory and populate it with the configuration files you need. See `Makefile.yp(8)` to learn which YP maps require which source files. For the format of the individual configuration files, refer to `passwd(5)`, `group(5)`, `hosts(5)` and so on, and look at the examples in `/etc`.

6. Create the initial version of your YP maps using the commands

```
cd /var/yp
make
```

Do not worry about error messages from `yppush(8)` right now. The YP server is not yet running.

7. YP uses `rpc(3)` (remote procedure calls) to communicate with clients, so it is necessary to enable `portmap(8)`. To do so, edit `rc.conf.local(8)` and set `portmap=YES`. This will start the portmapper on next boot. You can avoid rebooting by also starting it manually:

```
echo "portmap=YES" >> /etc/rc.conf.local
portmap
```

8. Consider using either the `securenet(5)` or the `ypserv.acl(5)` security feature of the YP server daemon. But be aware that both of these only provide IP based access control. Thus, they only help as long as potential attackers have neither physical access to the hardware of the network segments carrying your YP traffic nor root access to any host connected to those network segments.

9. Finally, start the YP server daemon:

```
ypserv
```

It will automatically be restarted at boot time as long as the directory `/var/yp/‘domainname‘` continues to exist.

10. To test the new server, consider making it its own client, following the instructions in the first part of the next section. In case you don't want the server to use its own maps, you can disable the client part after the test with the following commands:

```
pkill ypbind
rm -rf /var/yp/binding
```

11. If you wish to allow users to change their passwords from client machines, then you must enable `yppasswdd(8)`:

```
echo 'yppasswdd_flags="-d /etc/yp/src/puffynet"' >> /etc/rc.conf.local
rpc.yppasswdd
```

In case you left the source directory at the default `/etc`, just use `yppasswdd_flags=""`.

12. Remember that each time you change a file sourced by a YP map, you must regenerate your YP maps.

```
cd /var/yp
make
```

This updates all database files in `/var/yp/‘domainname‘`, with one exception: The file `ypservers.db`, listing all YP master and slave servers associated with the domain, is created directly from `ypinit -m` and modified exclusively by `ypinit -u`. In case you accidentally delete it, run `ypinit -u` to recreate it from scratch.

10.19.4 Setting up a YP client

Setting up a YP client involves two distinct parts. First, you must get the YP client daemon running, binding your client host to a YP server. Completing the following steps will allow you to retrieve data from the YP server, but that data will not yet be used by the system:

1. Like on the server, you must set the domain name and enable the portmapper:

```
echo "puffynet" > /etc/defaultdomain
domainname ‘cat /etc/defaultdomain‘
echo "portmap=YES" >> /etc/rc.conf.local
portmap
```

2. It is recommended to provide a list of YP servers in the configuration file `/etc/yp/‘domainname‘`. Otherwise, the YP client daemon will use network broadcasts to find YP servers for its domain. Explicitly specifying the servers is both more robust and marginally less open to attack. If you have not set up any slave servers, just put the host name of the master server into `/etc/yp/‘domainname‘`.

3. The YP client daemon is called `ypbind(8)`. Starting it manually will create the directory `/var/yp/binding`, such that it will be automatically restarted at boot time.

   ```
   ypbind
   ```

4. If all went well you should be able to query the YP server using `ypcat(1)` and see your passwd map returned.

   ```
   ypcat passwd
   bob:*:5001:5000:Bob Nuggets:/home/bob:/usr/local/bin/zsh
   ...
   ```

   Other useful tools for debugging your YP setup include `ypmatch(1)` and `yptest(8)`.

The second part of configuring a YP client involves editing local configuration files such that certain YP maps get used by various system facilities. Not all servers serve all standard maps supported by the operating system, some servers serve additional non-standard maps, and you are by no means compelled to use all those maps. Which of the available maps shall or shall not be used, and for which purposes they shall be used, is fully at the discretion of the client host's system administrator.

For a list of standard YP maps and their standard usage, see `Makefile.yp(8)`. The most common use cases include:

- If you want to include all user accounts from the YP domain, append the default YP marker to the master password file and rebuild the password database:

  ```
  echo '+:*::::::::' >> /etc/master.passwd
  pwd_mkdb -p /etc/master.passwd
  ```

  For details on selective inclusion and exclusion of user accounts, see `passwd(5)`. To test whether inclusion actually works, use the `id(1)` utility.

- If you want to include all groups from the YP domain, append the default YP marker to the group file:

  ```
  echo '+:*::' >> /etc/group
  ```

  For details on selective group inclusion, see `group(5)`.

- If you are distributing hostnames via YP, you should now review `resolv.conf(5)` and check that the name service lookup order is correct. Most users will require a line like this:

  ```
  lookup file yp bind
  ```

10.20 Character sets and localization

OpenBSD uses the ASCII character set by default. It also supports the Latin (ISO-8859-*), KOI-8, and Unicode (UTF-8) character sets.

10.20.1 Configuring the active character set

To use one of the extended character sets, the *LC_CTYPE* environment variable must be set to the name of a supported locale. *LC_CTYPE* will only affect the character set available to applications. It will not change the language used for application messages.

The list of supported locales can be obtained by running the command:
`ls /usr/share/locale`

The *LC_CTYPE* environment variable can be set in one of the following ways:

- If logging in via `xdm(1)` add a line such as
  `export LC_CTYPE="en_US.UTF-8"`
  to */.xsession* before starting the window manager (see the section on customizing X for details). This example enables the Unicode (UTF-8) character set and will also cause applications such as `xterm(1)` to enable UTF-8 mode by default.

- If logging in via the text console add a line such as
  `export LC_CTYPE="en_US.ISO8859-1"`
  to */.profile*. Note that the text console supports ASCII and ISO8859-1 only. It does not support UTF-8.

Few utilities in the base system support UTF-8 at this time. Most will use ASCII in the UTF-8 locale. However, many programs from the ports collection do support UTF-8.

UTF-8 can also be used with specific applications only by starting those applications in `uxterm(1)`. This works even if the login session uses a non-UTF-8 locale.

When logging into remote systems with `ssh(1)`, the *LC_CTYPE* environment variable is not propagated and will need to be manually set to the same value used by the local terminal.

10.20.2 Changing the language used in application messages

The language used for application messages can be changed by setting the *LC_MESSAGES* environment variable to the name of a supported locale. This can be done in the same way as described for *LC_CTYPE* above. Both *LC_MESSAGES* and *LC_CTYPE* should be set to the same value.

Few utilities in the base system support languages other than English at this time. However, many programs from the ports collection support localized

messages in various languages. They will fall back to English if the desired language is not available.

Chapter 11

The X Window System

11.1 Introduction to X

The X Window System (sometimes just called "X", other times, incorrectly called "X Windows") is the environment which provides graphics services to OpenBSD and other Unix-based systems. However, by itself, X provides very little: one also must have a "Window Manager", to present a user interface. Most of the "personality" one will feel from X will be due to the window manager, rather than X itself. OpenBSD ships with a free version of the `fvwm(1)` and `cwm(1)` window managers, although you may wish to use any of the other window managers that are in packages. Search using a key of "`window manager`" for a list of the many window managers available.

X is considered a "client-server" structured protocol, however the terminology is sometimes confusing. The computer with the graphics on the screen is the "X Server". The application which directs the X server what to put on its screen is the "X Client", even though it may be a much more powerful computer in a data center. This model can be used to have large, processor intensive applications (X clients) running on a very powerful machine, using the X Server running on a small, low power machine on your desk for their user interface.

It is possible to run X clients on a system without any graphical support. For example, one could have an application (the X client) running on an mvme88k, displaying its output on an alpha's graphical display (the X server). Since X is a well-defined, cross-platform protocol, it is even possible to have an X application running on (for example) a Solaris machine use an OpenBSD machine for its display.

The client and server can also be running on the same machine, and for most of this section, that will be the assumption.

11.1.1 How much computer do I need to run X?

X itself is a pretty large program, a fast computer will be appreciated if you are starting it and stopping it regularly. However, once running, it performs pretty responsively on a very modest computer. To get responsive display performance on some platforms, even for just text, you will want to run X. These platforms, such as sparc and sparc64, were intended to be used with a graphical interface, and the text console performance is very poor.

That being said, the point of running X is usually to run X applications. Some X applications are very lean, others will seemingly take and use all the processor and RAM you can give them. Of course, some users just like to use X to provide a large number of `xterm(1)`s, which can be done on very modest hardware.

11.1.2 Can I have any kind of graphics without X?

Assuming you won't accept ASCII graphics, that requires some kind of framebuffer console driver. Some operating systems provide this, but there is not

currently one for OpenBSD, nor is there much interest among developers for one.

11.2 Configuring X

Good news: In the vast majority of hardware in most platforms, X requires no configuration at all, it Just Works.

The details of manual configuration of X varies considerably from platform to platform. In all cases, there will be instructions and other platform-specific information in `/usr/X11R6/README` in the installed system.

Several platforms require the `xf86(4)` X aperture driver, which provides access to the memory and I/O ports of a VGA board and the PCI configuration registers required by the X servers. This driver must be enabled before it is used, either by answering "yes" to this question during install:

```
Do you expect to run the X window System
```

no

or by changing the value of `machdep.allowaperture` to the appropriate non-zero value in `/etc/sysctl.conf` for your platform, and rebooting the machine (this sysctl cannot be changed after boot has been completed for security reasons). There are security implications to this, so do not do this if you do not need it.

11.2.1 alpha

`/usr/X11R6/README` for alpha.

Set `machdep.allowaperture=1` in `/etc/sysctl.conf`.

The TGA and some VGA cards are supported. No further configuration should be needed.

11.2.2 amd64

`/usr/X11R6/README` for amd64

Set `machdep.allowaperture=2` in `/etc/sysctl.conf`.

For the vast majority of users, X on amd64 auto-configures successfully, so no further configuration is needed. IF further configuration is needed, use the X -configure process below.

11.2.3 armish

No X servers, only X clients.

11.2.4 hp300

`/usr/X11R6/README` for hp300

11.2.5 hppa

No X server, only X clients.

11.2.6 i386

`/usr/X11R6/README` for i386.

Set `machdep.allowaperture=2` in `/etc/sysctl.conf`.

For the vast majority of users, X on i386 auto-configures successfully, so no further configuration is needed. IF further configuration is needed, use the X -configure process below.

11.2.7 landisk

No X servers, only X clients.

11.2.8 loongson

No configuration needed.

11.2.9 luna88k

No X servers, only X clients.

11.2.10 macppc

`/usr/X11R6/README` for macppc

Set `machdep.allowaperture=2` in `/etc/sysctl.conf`.

Supported Macintosh PPC systems can be run in one of two different ways: "accelerated" and "framebuffer" (unaccelerated).

In the "framebuffer" mode, the system will be running with 8 bits per pixel, and the video resolution is controlled by the Macintosh environment, so you will probably want to keep a small MacOS section on your disk to adjust these settings. This mode has the advantage of "Just Working", however it can be frustratingly inflexible (for example, altering resolution may require booting MacOS).

If your Macintosh has an ATI-based video system, it can run using an accelerated X server, which gives better performance and more control in the OpenBSD environment. The NVIDIA video cards in some macppc systems will also work in many cases. The README file has details on configuring the accelerated driver, start by using the sample file there.

While the README file details using the standard Apple one-button mouse with X, unless you are using a laptop, it is highly recommended that you just buy a modern third-party USB mouse with three or more buttons.

CRT iMacs and X

iMacs have a very unusual (in this day) CRT, in that it has a fixed horizontal scan frequency. Attempting to use a horizontal sweep speed outside a very narrow range will cause the monitor to not light up. The following lines, added to xorg.conf in the `Section "Monitor"` area seem to make a lot of CRT-based iMacs in the 333MHz to 500MHz range (and possibly more!) work well:

```
HorizSync       60.0 - 60.0
        VertRefresh     43.0 - 117.0
```

You may wish to restrict the lower limit of `VertRefresh` to values that you are more likely to find acceptable, for example 70.

11.2.11 mvme68k

No X servers, only X clients.

11.2.12 mvme88k

No X servers, only X clients.

11.2.13 sgi

X runs on the O2 system's frame buffer in unaccelerated mode.

11.2.14 sparc

`/usr/X11R6/README` for sparc.

With a single supported framebuffer, there is no configuration needed. If you wish to use a multi-headed configuration, see the above README file for details.

Resolution is controlled by the firmware before booting OpenBSD

11.2.15 sparc64

`/usr/X11R6/README` for sparc64.

Most simple configurations now "Just Work". If yours does not or you wish to get fancy, see the above README file.

11.2.16 vax

`/usr/X11R6/README` for vax

The X server currently only works on VAXstation 4000 models with either a `lcg(4)` or `lcspx(4)` framebuffer.

11.2.17 zaurus

`/usr/X11R6/README` for zaurus.

No configuration needed, X "Just Works".

11.3 Configuring X on amd64 and i386

If you haven't tried just starting X, stop! Don't make work for yourself if you you don't need to! Most i386 and amd64 users will find X Just Works without any manual configuration.

11.3.1 X.Org configuration

Sometimes, X does not "just work" as desired, and sometimes, you need to customize something that does work.

If X does not work as desired with your system, you will need to create a configuration file. A good starting place (and sometimes, a good ending place!) is to run `Xorg(1)` in the "X -configure" mode. This will load all available video driver modules, probes for other hardware, and based on what it finds, writes out a xorg.conf file that may or may not work. Even if it does not work, it will be a useful starting point for creating one that works as desired.

Another time-honored way to configure X is to use your favorite search engine to hunt for someone else who already solved your problem. Note that `xorg.conf` files for other Unix-like OSs will often provide very useful tips on what you need to get X running on your similar hardware. While that's not a bad way, that method won't be emphasized here.

11.3.2 Our example machine

As a demonstration of setting up X, we will use an old Celeron 400MHz system, with an AGP video slot. The video card is an old AGP card, shown in the dmesg as:

```
vga1 at pci1 dev 0 function 0 "3DFX Interactive Banshee" rev 0x03
```

This is a once high-end video card, with 16M RAM, but is now mostly unsupported on modern versions of "mainstream" operating systems. The monitor attached to the system is a Sony Multiscan G400 19" CRT monitor, and it would be nice to run this monitor at 1280x1024, with a decent refresh rate, and 24 bit color.

First, after installing OpenBSD with X (and making sure the aperture driver is enabled in the kernel), let's see what X.Org's auto detection and configuration does, after all, we might get lucky. So, we simply log in and use the command `startx(1)`. The screen goes blank for a few moments, then we get the X "checkerboard" background, the "X" cursor, and then an xterm window.

It works!

More or less. While the system is fully functional, it doesn't appear to have picked up any of the capabilities of the monitor, and is running at what is clearly a low resolution (640x480). We hope to do better than this. Much better, in fact. This does mean we need to make our own xorg.conf file, however.

Let's use the "`X -configure`" process to generate a starting `xorg.conf` file. You will need to do this as root:

```
# X -configure
 [...]
Your xorg.conf file is /root/xorg.conf.new

To test the server, run 'X -config /root/xorg.conf.new'
```

By the way, the message is serious – use the entire path to your `xorg.conf.new` file, even if it is sitting in your current default directory. Failing to do so will result in `X(7)` not finding the file, and it will silently use the default configuration, and if that had worked, you wouldn't be using this process! This can set back your troubleshooting quite a bit. Trust us on this.

Let's do as it says, and see what we get:

```
# X -config /root/xorg.conf.new
```

Now, all we get is a black screen. Things had started out so well...

This might be a really good time to talk about ways of exiting X when started in this way. In order of preference:

- **CTRL-ALT-Backspace**: This hopefully causes X to immediately terminate, along with all X applications that are running. Of course, during the configuration process, you don't have any applications running, so this is not a problem (and in fact, at this point, this is your best way to exit X).
- **SSH into the box**, and "pkill Xorg", which may kill the X process, and may return you to a usable console.
- **SSH into the box** and reboot it.
- **Reset or power button**. Sometimes things go really bad. Yes, it is usually good to get X running before you load critical applications on the system. Sometimes, a bad X configuration will hang the entire machine to the point that only a hard reset will resolve the problem.

Fortunately for us, CTRL-ALT-Backspace does the job here, and we are returned to a command prompt. So now we need to see if we can figure out

what is wrong. First, we should look at what Xorg thinks is going on, and that is recorded in the file `/var/log/Xorg.0.log`. In this case, it appears that X thinks all is running fine, there are no obviously significant errors shown in the log (lines that start with an "`(EE)`" are errors).

Here is where knowing your hardware comes in handy. Attaching this system to a different monitor while it is showing the blank screen produces a "Sync. Out of Range" message on the display. So, apparently the configuration X gave us will not run on this monitor, and may not run on ANY monitor, if a video mode was selected that isn't possible for this particular card (keep in mind, X is looking at the chips on the card, and what they are potentially capable of, not how the manufacturer put it all together). Different monitors will do different things when the timing is way off, some will attempt to display what they can, others will drop to power saving mode, some will make horrible noises, some will display useful messages on the screen. This monitor seems to do none of the above. A note is made to NOT use this monitor for initial X configuration in the future.

Looking through the generated xorg.conf.new file, we see this:

```
Section "Monitor"
        #DisplaySize       370   270     # mm
        Identifier   "Monitor0"
        VendorName   "SNY"
        ModelName    "SONY CPD-G400"
 ### Comment all HorizSync and VertSync values to use DDC:
        HorizSync    30.0 - 107.0
        VertRefresh  48.0 - 120.0
        Option      "DPMS"
EndSection
```

As a test, let's try using DDC ("Data Display Channel", a way the monitor can tell the computer and video card what it is capable of), and see what happens. This time, we get the X mesh pattern and the moving cursor, which is all we expect when invoking X in this way (we terminate X using the CTRL-ALT-Backspace trick above). It is (again) a very low resolution, but it is working, so we can be pretty sure we have a timing and resolution problem. We'll restore the "HorizSync" and "VertRefresh" lines as they were, as we have verified this monitor's specs through a bit of Internet searching.

Let's try to force Xorg to a particular resolution, and see if we have any luck. In the `Section "Screen"` part of the `xorg.conf` file, we want to add a couple lines. The added lines are shown in bold:

```
Section "Screen"
        Identifier "Screen0"
        Device     "Card0"
        Monitor    "Monitor0"
        DefaultDepth   24
```

```
        SubSection "Display"
                Viewport   0 0
                Depth     1
        EndSubSection
        SubSection "Display"
                Viewport   0 0
                Depth     4
        EndSubSection
        SubSection "Display"
                Viewport   0 0
                Depth     8
        EndSubSection
        SubSection "Display"
                Viewport   0 0
                Depth     15
        EndSubSection
        SubSection "Display"
                Viewport   0 0
                Depth     16
        EndSubSection
        SubSection "Display"
                Viewport   0 0
                Depth     24
                Modes "1280x1024"
        EndSubSection
EndSection
```

These two changes tell X we want to use a 24 bit display depth, and for 24 bit depths, we want the resolution 1280x1024. As no other resolution is listed for "Depth 24", the system will be forced to that resolution.

We test as above, and... SUCCESS! We have what appears to be a very nice, high resolution display. Note that ALL that is expected is a mesh pattern (very good for seeing how good your monitor REALLY is and also great for calibrating LCD displays, called the "root weave") and a movable cursor. It is not intended to be functional at this point.

Now, we want to install this xorg.conf file so we can see how well we are really doing with usable running of X.

```
# cp xorg.conf.new /etc/X11/xorg.conf
```

We can now try X using the normal `startx(1)` command. It works!

It would probably be good to verify that we are at the resolution and color depth we desire, and also that we are running at a respectable refresh rate. We can do that with the `xrandr(1)` and `xdpyinfo(1)` commands. Among other things, `xdpyinfo(1)` tells us:

...

```
screen #0:
  print screen:    no
  dimensions:    1280x1024 pixels (433x347 millimeters)
  resolution:    75x75 dots per inch
  depths (7):    24, 1, 4, 8, 15, 16, 32
  root window id:    0x44
  depth of root window:    24 planes
```

...

So, yes, we are running at 1280x1024 with a depth of 24 planes (bits). `xrandr(4)` tells us:

```
SZ:    Pixels          Physical       Refresh
*0   1280 x 1024   ( 433mm x 347mm )  *85   75   60
 1   1280 x 960    ( 433mm x 347mm )   85   60
```

...

which tells us we are running with an 85Hz refresh rate, so this should be a very comfortable setting for most users.

11.3.3 What if it isn't that "easy"?

Sometimes, things just don't go together. Here are some tips.

- Read the man page for the X server you are using. In our example, `/var/log/Xorg.0.log` shows X is using TDFX as the driver, so that would be the `tdfx(4)` man page. You will often find tips, limitations, and options for configuring your video card. These vary from driver to driver, so don't assume you don't need to read the man page for the driver you are using now because you read a different one before.
- Try different monitors. As we discovered in our above example, different monitors will often give different clues as to what might be wrong.
- Try the `vesa(4)` X driver. This is definitely a "last choice" for performance reasons, but it works on almost all video cards, including those for which none of the "better" X server drivers will work with.
- Use different hardware. If you have a choice of video cards to use, try some others.

11.4 Starting X

There are two common ways to run X:

11.4.1 On Demand:

Log in to a console as normal, then run `startx(1)`.

11.4.2 Boot directly into X:

This is done using `xdm(1)`, the X Display Manager. xdm(1) is started as root, normally by `rc`, and presents a login prompt. Upon successful login, it starts an X session for that user. If or when that X session should be terminated (including by hitting CTRL-ALT-Backspace), `xdm(1)` will return, prompting the user to login again. For this reason, do NOT attempt to start `xdm(1)` from `/etc/rc.conf.local` until you have verified X works as you wish, or your machine may become very difficult to manage! (worst case: boot single user, as if you lost your password, and edit out the `xdm_flags` line in your `/etc/rc.conf.local` file.)

On some platforms, you will need to disable the console `getty(8)` to use `xdm(1)`.

11.5 Customizing X

11.5.1 Introduction

OpenBSD's default X environment is fully functional, but you may wish to customize it. You may wish to change the background pattern or color, or you may wish to change the Window Manager (the program that most defines your X environment), or change the applications that are started when X starts.

The default window manager in OpenBSD is `fvwm(1)`. Fvwm is a good, general purpose window manager, but it is hardly your only choice; it isn't even the only window manager included with OpenBSD (see `cwm(1)` and `twm(1)`). A large number of window managers are also available through packages.

Similar to the system startup script, X has a process it goes through to set up the user environment. More accurately, it has more than one process; which process is used depends on how you start X. Understanding how X starts will help you understand how to customize your work environment the way you wish it to be.

Note that you can customize the environment on both a system-wide and user level. It is probably best to do user level changes rather than to change the system defaults, as the user scripts are stored in the user's home directory, so you will have less merging of files to do when upgrading to a new version of OpenBSD. The system-wide defaults are in `/etc/X11` and were initially loaded from `xetcXX.tgz`, which is not reloaded by the suggested upgrade process, so if you make system-wide changes, they will persist, but you may need to merge those changes into later versions of those files.

11.5.2 `startx(1)` startup

`startx(1)` looks for the file `.xinitrc` in the user's home directory. .xinitrc is usually a shell script, which can start as many X "client" (applications that use X) programs as desired. When this script exits, the X server shuts down. Generally, most of the programs run by this script should run in the background, though the last one should run in the foreground (typically the window manager); when it exits, the script will exit, and X will be shutdown.

In the simplest case, this can be as little as just the name of the window manager you wish to invoke:

```
cwm
```

Or you can get a little more fancy:

```
xconsole -geometry -0+0 -fn 5x7 &
oclock -geometry 75x75-0-0 &
xsetroot -solid grey &
cwm
```

That will start the `xconsole(1)` which provides a copy of any text that the kernel would have sent to the console (which is now covered by the graphical screen), an analog clock, `oclock(1)`, and sets the background to a solid grey background with `xsetroot(1)` all before invoking the `cwm(1)` window manager. Note that only the window manager is not "backgrounded" with an "&" character. This means that X will stay running until `cwm(1)` exits.

If a user's home directory does not have a `.xinitrc` file in it, the system's `/etc/X11/xinit/xinitrc` file is used. This file can provide you some additional ideas for your `.xinitrc` script.

11.5.3 `xdm(1)` startup

`xdm(1)` is usually started by the system startup scripts, but for testing purposes (recommended, until you know your have your X config right!), it can be run as root.

`xdm(1)` has a lot of other functionality that won't be touched on here, but for our purposes, xdm will present the user with a login screen, get and verify a user name and password, and then give the user their X environment. When X shuts down, either deliberately or accidently, xdm will start it back up again. This is why you want to make sure X is configured properly before using `xdm(1)`, and certainly before having `xdm(1)` start at boot, otherwise, you may have some difficulty getting control of the machine.

When `xdm(1)` starts, it runs the `/etc/X11/xdm/Xsession`, which will check to see if the user has a `.xsession` file in their home directory. So, if you wish to change your default window manager, simply invoke it (and maybe other things) in `.xsession`. Again, any programs you want started with X (for example, maybe three `xterm(1)`s) can be placed here, but all should be backgrounded

except for your window manager, as again, when that exits, your X session will be ended. In this case, `xdm(1)` will restart X and bring you back to a login screen.

11.5.4 Trying a new window manager

You can invoke a particular window manager when you load X without altering any defaults like this:

```
$ startx /usr/local/bin/fluxbox
```

Several window managers (including `cwm(1)` and `fvwm(1)`) offer the ability to change window managers on the fly, without restarting X or any of your applications. Your new window manager replaces your old one; exiting the newly-loaded window manager terminates X, it does not return you back to your previous window manager. `fvwm(1)` allows you to start a different window manager by left clicking on the background ("root window"), chose "(Re)Start", then pick your preferred window manager (however, note that you will need to add your alternative window managers to your `.fvwmrc` file (the system-wide default is `/usr/X11R6/lib/X11/fvwm/.fvwmrc`)). `cwm(1)` allows you to invoke another window manager by hitting Ctrl-Alt-w, and typing in the manager you wish to switch to.

Once you have found a window manager you like, you can set it as the final program run by your startup scripts as described above.

Chapter 12

Hardware and Platform-Specific Questions

12.1 General hardware notes

12.1.1 PCI devices

- PCI devices are mostly self-configuring – the computer and OS will allocate resources to the cards as required.
- Interrupts can be shared on the PCI bus. Not only can they be, the system will often perform better when the IRQs are shared, especially on i386 systems.
- There are several different PCI bus standards. You will occasionally find a PCI2.2 specification card that will just not work in a PCI2.1 specification system. Also, many cards with on-board bridges (such as, multi-port network cards) will not work well in older systems.
- The PCI bus supports two levels of signaling, 3.3V and 5V. Cards that work with 3.3V signaling have a second notch cut in their PCI connector. Most PCI cards use 5V signaling, which is used by most computers. The Soekris single-board computers (Net45x1 and Net4801) are commonly-encountered computers that only support 3.3V signaling.

12.1.2 ISA devices

- ISA devices cannot share resources, and in general, must be manually configured to settings that don't conflict with other devices in the system.
- Some ISA devices are "Plug and Play" (isapnp(4)) – if you have any problem with these devices, though, verify their configuration in your dmesg(8), ISAPnP doesn't always work as desired.
- In general, if you have a choice, most people are best advised to avoid ISA cards in favor of PCI. ISA cards are more difficult to configure and have a much greater negative impact on the system's performance.

12.1.3 My device is "recognized" but says "`not configured`" in dmesg

In short, it means your device is not supported by the kernel you are using, so you will not be able to use it.

PCI and many other types of devices offer identifying information so that the OS can properly recognize and support devices. Adding recognition is easy, adding support is often not. Here is part of a dmesg with two examples of "not configured" devices:

```
...
vendor "Intel", unknown product 0x5201 (class network subclass ethernet,
rev 0x03) at pci2 dev 9 function 1 not configured
```

```
...
"Intel EE Pro 100" rev 0x03 at pci2 dev 10 function 0 not configured
...
```

The first one (a network adapter) had its vendor code identified and the general type of card was determined, but not the precise model of the card. The second example was another network adapter, this one a developer had seen and had entered into the identification file that is used to identify the card. In both cases, however, the cards will be non-functional, as both are shown as "`not configured`", meaning no driver has attached to the card.

What can I do about a `not configured` device?

- If the device or card you are seeing is not one you need, you can safely ignore the "not configured" devices, they will not hurt your system. Some "special purpose" devices are deliberately left unconfigured so the system's BIOS will handle them.
- In some cases, it is just a variation of an already supported device, in which case, it may be relatively easy for a developer to add support for the new card. In other cases, it may be a totally unsupported chip set or implementation (such as the above examples). In that case, a new driver would have to be written, which may not even be possible if the device is not fully documented. You are certainly welcome to write a driver for the device yourself.
- If you are running an install kernel, the device may not be supported by the install media you used, but may be supported by a different boot disk. This is a common issue with users of some popular SCSI cards who misread the footnotes on the i386 platform page and try all the boot floppies their SCSI card is NOT supported on, rather than the one that it is supported on.
- If you are running a modified kernel, you may have removed support for a device you now need. In general, removing devices from a kernel is a bad idea. This is one reason why.
- Before reporting a "not configured" device, make sure you have first tested the most recent snapshot, as support may already have been added, and check the mail list archives to see if the issue has been discussed already. Remember, however, if you are using an older version of OpenBSD, you will generally have to upgrade to get the benefit of any new driver written.

12.1.4 I have a card listed as "supported", but it doesn't work!

Unfortunately, many manufacturers use product model numbers to indicate marketplace position, rather than the technical nature of a product. For this rea-

son, you may buy a product with the same name or model number as a product listed in the platform pages, but end up with a totally different product that may not work with OpenBSD. For example, many early wireless network adapters were based on the Prism2 chip set, using the (`wi(4)`) driver, but later, when lower-cost chips became available, many manufacturers changed their product to use chips for which no open source drivers exist, but never changed their model numbers. Wireless network adapters, unfortunately, are far from the only example of this.

12.1.5 Are WinModems supported?

WinModems are low-cost modems which rely on the processor to do much of the signal processing normally done in hardware in a "real" modem. Due to the variety of incompatible and typically undocumented WinModem chips, there is no support for WinModems in OpenBSD, and this is not likely to change.

12.1.6 What happened to the Adaptec RAID support (`aac`)?

Adaptec has refused to provide useful and accurate documentation about their FSA-based (`aac(4)`) RAID controllers. As these RAID controllers seem to be very buggy, this documentation is critical for a useful driver. Since this driver was so unreliable, it was removed from the GENERIC kernel.

I can compile my own kernel with `aac(4)` support, right?

Sure. But what part of "unreliable" did you fail to understand? This isn't an "experimental" feature, this is a known-flawed driver. Maybe it works with some variations of hardware sufficiently well to be usable, but we don't recommend betting your data on it.

12.1.7 My `ami(4)` card will only support one logical disk!

There is a known bug with `ami(4)` which will cause data corruption if you use more than one volume on some controllers. On controllers with this issue, OpenBSD will limit you to one logical disk, resulting in a message in your dmesg that looks like:

```
ami0: FW A.04.03, BIOS vA.04.03, 4MB RAM
ami0: 3 channels, 16 targets, 2 logical drives
ami0: firmware buggy, limiting access to first logical disk
scsibus0 at ami0: 1 targets
```

12.1.8 How do I activate my crypto accelerator card?

Supported crypto accelerator hardware, such as the `hifn(4)` based cards will "just work" when installed in an OpenBSD system. No activation or other configuration is needed.

The overhead of talking to one of these devices will sometimes negate much or all of the benefit of offloading the processor of the crypto tasks. Your results may vary widely depending on the task you have to accomplish.

12.2 DEC Alpha

nothingyet

12.3 AMD 64

12.3.1 Can I run OpenBSD/amd64 on my Intel P4 or AMD Sempron?

In the case of many recent processors, the answer is "yes". Unfortunately, trying to find out which processor variations do and which do not properly support the amd64 instruction set is difficult. It is usually easier to just try it and see if it works.

Processors that do not support the amd64 instruction set will fail very early when booting the kernel; there will be no question about "does it work?" in your mind.

12.3.2 Can I run my i386 binary on OpenBSD/amd64?

No.

There are a number of reasons one may desire to use OpenBSD/i386 over OpenBSD/amd64, even on hardware that supports amd64 code:

- Need for i386 binary (or other OS) compatibility.
- Need to run applications that are not "64 bit clean".
- Need for ability to move disks to another machine that isn't amd64 capable
- For some applications and on some hardware, OpenBSD/i386 may outperform OpenBSD/amd64. Relatively few people will ever experience this, and work is on-going to eliminate as many of these situations as possible.

12.4 ARM-based appliances

[nothing yet]

12.5 HP300

[nothing yet]

12.6 HPPA

[nothing yet]

12.7 i386

12.7.1 ISA NICs

As OpenBSD runs well on older hardware, users often will end up using ISA NICs on OpenBSD systems. ISA hardware requires much more configuration and understanding than does PCI hardware. In general, you can't just stuff the card in the computer and expect it to magically work. In many machines, if your ISA device is not in a "Plug 'n' Play" (PNP) mode, you must reserve the resources the card uses in the system's BIOS.

3Com 3C509B `ep(4)`

This is an excellent performing ISA NIC, supported by the `ep(4)` driver. The 'B' version can be distinguished from the non-B version by labeling on the card and by the larger "main" chip on the board (approximately 2.5cm on a side for the 'B' version, vs. 2cm on a side on the older version), and will provide better performance on a loaded or dual network card system. The 3C509B ships configured in a PNP mode, which unfortunately does not comply with standards, and causes problems in OpenBSD's `isapnp(4)` support. The adapter is picked up first as a non-PNP device, then again after the PNP support comes on-line, resulting in an extra NIC showing in the dmesg. This may work fine, or it may cause other problems. It is highly recommended that the 3C509B cards have PNP mode disabled and manually configured to non-conflicting settings using the 3Com DOS-based configuration utilities before configuration.

The `ep(4)` driver will pick the cards up at any hardware combination that does not conflict with other devices in the system.

If you have multiple 3C509 cards in your system, it is recommended that you label the cards' spine with the MAC address, and use the dmesg to identify which is which.

Note that the 3C509, the 3C905 and the 3C590 are often confused. The 3C509 is a 10Mbps ISA card, the 3C905 and 3C590 are PCI cards.

NE2000

The original NE2000 NIC was developed in the mid-1980s by Novell. Since then, many manufacturers have produced cards that are very similar, which are generally called NE2000-compatibles, or clones. Performance of these clone cards varies greatly. While some older NE2000-compatible cards performed very well, many of the currently-available ones perform poorly. NE2000-compatibles are supported by the `ne(4)` driver in OpenBSD.

OpenBSD will handle some ISAPNP-capable NE2000-compatible cards well if the ISAPNP mode is turned on. Other cards will have to be set using either jumpers or a DOS-based configuration utility. Unfortunately, as the original NE2000 cards did not have software configuration or ISAPNP support, there are no standards for this – you need the utility that will have been originally supplied with your specific card. This can often be difficult to obtain.

The `ne(4)` driver supports three configurations of the ISA NE2000 card in the GENERIC OpenBSD kernel:

```
ne0:  port 0x240 irq 9
     ne1:  port 0x300 irq 10
     ne2:  port 0x280 irq 9
```

If these settings are not acceptable, you can adjust them using User Kernel Configuration (UKC) or by building a customized kernel.

Note that the `ne(4)` driver is fairly "dumb" – only the I/O port is probed, if any of the above I/O addresses is detected, the corresponding IRQ is *assumed*. `dmesg(8)` will not reflect the actual IRQ of the adapter in the case of ISA `ne(4)` drivers. If this is not the actual IRQ your card is set to, it will not work.

Note that there are non-ISA cards that use the `ne(4)` driver – PCI and PCMCIA `ne(4)` cards exist. These notes do not apply to them, these devices are auto-configuring.

12.7.2 OpenBSD won't work on my 80386/80386SX/80486SX system!

80386SX/DX

Support for the 80386DX and 80386SX processors was dropped beginning with OpenBSD 4.2. In addition to limitations of the 80386 chip, the systems are just too slow and rarely had enough RAM and a required FPU to run OpenBSD.

80486SX

The 80486SX chip was a "low-cost" version of the 80486, which lacked the hardware floating point support (like the 80386) OpenBSD requires. Fortunately, full 80486DX chips are fairly available, and is an easy upgrade in most systems.

The 80486DX and newer chips run OpenBSD fine.

12.7.3 My dmesg shows multiple devices sharing the same Interrupt (IRQ)!

This is entirely acceptable, and in fact, even desirable for PCI devices. This is a design feature of the PCI bus. Some people will say that sharing interrupt requests (IRQs) is bad, however they are either confusing the situation with the ISA bus (where IRQ sharing is not permitted), or past experience with broken hardware or software.

ISA devices can not share IRQs. If you find ISA devices sharing IRQs, you must correct this problem.

12.7.4 My keyboard/mouse keeps locking up (or goes crazy)!

This is most often seen when using a "switch box" (sometimes called a "KVM switch") to attach multiple computers to one keyboard, monitor and mouse. You can experiment with different brand and design switch boxes, but OpenBSD seems to be more sensitive to switching the mouse than some other operating systems. The problem is usually just the switching of the mouse. If you are not using the mouse, the solution is simple: don't attach the mouse cable to the computer. If you are using the mouse, an easy solution is "one mouse per computer", and switch just the keyboard and monitor. You may find using a PS/2 Mouse -¿ USB port adapter (so OpenBSD sees a USB mouse) will work around this problem. If you just want console access to the machine, you may wish to consider using a serial console instead.

12.7.5 My Soekris performs poorly

The Soekris machines are nice, low-power, small and modest cost PC class systems, and many OpenBSD users are quite pleased with them. However, it must be remembered the processors, NICs, and other devices on the system are all chosen for low power consumption, not performance. While they are suitable for many applications, they are not high performance machines.

12.7.6 I get "missing interrupt" errors on my CF device

With some CF systems, such as early Soekris 4801 systems and some CF adapters, the DMA support is not properly wired up, so you might see messages such as:

```
pciide0:0:0: bus-master DMA error: missing interrupt, status=0x20
```

In these cases, you will need to disable DMA in the `wd(4)` driver to avoid this problem. This can be done using ukc to install, then config(8) to change it permanently:

```
ukc> change wd
 42 wd* at wdc*|pciide* channel -1 flags 0x0
change [n] y
channel [-1] ?
flags [0] ? 0x0ff0
 42 wd* changed
 42 wd* at wdc*|pciide* channel -1 flags 0xff0
ukc> quit
```

12.7.7 Is this really new CPU supported?

The issue of support is rarely the CPU itself, it is almost always the surrounding and supporting hardware.

Most likely, yes, the latest CPU is supported just fine. The surrounding support hardware may not work if it is significantly different than earlier versions. If you have problems with new hardware, make sure you try a snapshot before making a bug report.

12.8 Landisk

[nothing yet]

12.9 Luna88k

[nothing yet]

12.10 MacPPC

12.10.1 My `bm(4)` network adapter doesn't work!

The bm driver, supporting the BMAC chip used on some MacPPC systems (including early iMacs) has issues. A usb NIC is recommended for these systems at this time.

12.11 MVME68k

[nothing yet]

12.12 SGI

[nothing yet]

12.13 SPARC

[nothing yet]

12.14 UltraSPARC (sparc64)

12.14.1 My UltraSPARC won't boot from the floppy image

Only the Ultra 1/1e and Ultra 2 can boot any OS from floppy disk. Use CD-ROM, Miniroot, or network boot to do your installation instead.

12.14.2 I'm getting "partition extends past end of unit" messages in disklabel

On sparc and sparc64 systems, the BSD disklabel cannot describe a disk geometry larger than 8GB, while individual disklabel entries can be larger.

Everytime you run `disklabel(8)`, it performs some sanity checks of the disklabel entries against what it thinks is the correct drive geometry, and since it sees a truncated geometry, it warns and will not let you edit entries outside this 8GB area unless you tell it to use the real drive geometry. Do this using the 'g' command of the command-driven editor of `disklabel(8)` and tell it to use the "

d

isk geometry":

```
# disklabel -E wd0
# Inside MBR partition 3: type A6 start 63 size 17912412
   [...]
Initial label editor (enter '?' for help at any prompt)
> g
[d]isk, [b]ios, or [u]ser geometry: [d] d
> w
> q
```

You will still get the warning messages, but you can configure and use your disk as desired. A proper solution would have to be compatible with existing systems already in use, plus be compatible with Solaris running on disks larger than 8GB, but this has not been worked out yet.

12.15 DEC VAX

12.15.1 Can I use the SIMH VAX simulator?

Yes!

The SIMH VAX simulator can be used to effectively emulate a real VAX. Instructions can be found in the OpenBSD/vax on SIMH page.

12.16 Sharp Zaurus

12.16.1 USB devices aren't working properly

The Zaurus has very little current available on its USB port, so many USB devices will not work if they are directly attached to it. You will need to use a powered USB hub to run these devices.

Chapter 13

Multimedia

13.1 How do I configure my audio device?

The devices in OpenBSD that are related to audio are: `/dev/audio`, `/dev/sound`, `/dev/audioctl` and `/dev/mixer`. For a good overview of the audio driver layer, please read the `audio(4)` manual page.

All supported audio drivers are already included in the GENERIC kernel so there is no need for extra configuration or installation of drivers. To find out about options for your specific sound chip, you must find out which sound chip you have. Supported chips may be found on the hardware compatibility page for your platform. When you already have OpenBSD running, look for a sound driver in the output of the `dmesg(8)` command, and read its manual page to find more specific information like options and other details about the driver. An example of an audio chip in a dmesg output is:

```
auich0 at pci0 dev 31 function 5 "Intel 82801BA AC97" rev 0x04: irq 10, ICH2 AC97
ac97: codec id 0x41445360 (Analog Devices AD1885)
ac97: codec features headphone, Analog Devices Phat Stereo
audio0 at auich0
```

OpenBSD base provides two tools for monitoring and configuring audio devices. `audioctl(1)` is used for the audio processing parameters, such as encoding, sample rate and number of channels, while `mixerctl(1)` is used for the mixing parameters, such as channel source, gain level and mute.

The following command uses `audioctl(1)` to display the default processing parameters of an audio device.

```
$ audioctl -f /dev/audio
...
```

Note that `-f /dev/audio` was used explicitly. Opening `/dev/audio` causes the audio device to reset to the default parameters, which is what we wanted to see.

`audioctl(1)` is also quite useful for exploring the capabilities of an audio device. For example, to see if the device supports some common sample rates, you could simply try setting the playback rate:

```
$ audioctl play.rate=48000
play.rate: -> 48000
$ audioctl play.rate=44100
play.rate: -> 44100
$ audioctl play.rate=22050
audioctl: set failed: Invalid argument
$ audioctl play.rate=8000
audioctl: set failed: Invalid argument
$
```

This device supports 48000 and 44100 Hz playback rates, but not 22050 or 8000. Note that if a sample rate is not supported, there is not always an error message, but the returned sample rate is not the one that was desired.

```
$ audioctl play.rate=48000
play.rate: -> 48000
$ audioctl play.rate=44100
play.rate: -> 48000
$ audioctl play.rate=22050
play.rate: -> 48000
$ audioctl play.rate=8000
play.rate: -> 48000
$
```

This device supports 48000 Hz playback only.

Audio hardware is usually capable of at least some minimal mixing. Running `mixerctl(1)` with no arguments will list the device's mixer controls and current settings.

```
$ mixerctl
...
```

Some devices have only a handful of controls, some have a hundred or more. Note that not every option of every audio chip necessarily reaches the outside world. So there may be, for example, more outputs listed than are physically available on a sound card or motherboard.

There are a few controls that are common to many devices:

- `outputs.master` controls the playback output level
- `inputs.dac` controls the level from the DAC (digital to analog converter), used when playing an audio file
- `record.source` controls what inputs are mixed into the ADC (analog to digital converter), used when recording
- `record.volume` or `record.record` controls the input level of the ADC

The controls of an audio device may be labeled differently. For instance, there might not be an `outputs.master` as above, but there is an `outputs.outputs` which does the same thing. Usually the controls have a meaningful label, but sometimes one must simply try different settings to see what effect each control has.

Some devices use what is known as EAPD, which stands for external amplifier power down. However, this is just another on/off switch. It is probably refered to as "power down" because it is often used for power saving, which means this type of control is more often found in laptops. Sometimes it is necessary to set controls with eapd or extamp in their name to on to get an output signal.

As a basic example of common mixerctl usage, to set the volume of the left and right channels to 200, you would issue

```
$ mixerctl outputs.master=200,200
outputs.master: 255,255 -> 207,207
```

Notice how the value becomes 207. The reason for this is that this audio device has an AC'97 codec, which uses only 5 bits for volume control, leading to only 32 possible values. Other hardware could have different resolution.

To unmute the master channel, you would do

```
$ mixerctl outputs.master.mute=off
outputs.master.mute: on -> off
```

To make the changes take affect on each reboot, edit */etc/mixerctl.conf*, for example:

```
$ cat /etc/mixerctl.conf
outputs.master=200,200
outputs.master.mute=off
outputs.headphones=160,160
outputs.headphones.mute=off
```

13.2 Playing different kinds of audio

Digitized audio

Lossless audio formats (AU, PCM, WAV, FLAC, TTA)

Some of the lossless audio formats may be played without the need for third party software, provided they contain the uncompressed digital samples in chunks of bytes. These formats include Sun audio (AU), raw PCM files (without headers), and RIFF WAV.

OpenBSD comes with `aucat(1)`, a program for recording and playing uncompressed audio. The following example will play a WAV file.

```
$ aucat -i filename.wav
```

`aucat(1)` supports both headerless and WAV audio files with the -i option. aucat also plays Sun audio files where the audio data is encoded as 8 kHz monaural mulaw, which is the most common encoding for this type of audio file.

It is also possible to play uncompressed audio data by passing it directly to the audio device. To do this, you need to know its main parameters: encoding type, number of channels, sample rate, bits per sample. If you don't know this, you might find out with the `file(1)` utility:

```
$ file music.au
music.au:  Sun/NeXT audio data: 16-bit linear PCM, stereo, 44100 Hz
$ file music.wav
music.wav: Microsoft RIFF, WAVE audio data, 16 bit, stereo 44100 Hz
```

The only remaining things to know about these example files is that they use little-endian byte ordering and signed linear quantization. You could figure this out by reading the header with **hexdump(1)**. If you are using a headerless (raw) file, there is no way to know the parameters beforehand. Set the following parameters accordingly using **audioctl(1)**.

```
play.encoding=slinear_le
play.rate=44100
play.channels=2
play.precision=16
```

Next, pass the audio file to the sound device:

```
$ cat music.au > /dev/sound
```

If you applied the correct settings, you should be hearing what you expected.

Note: Always use `/dev/sound`, not `/dev/audio`, if you want the settings you applied with audioctl to stay in place.

There are, of course, other utilities you can use to play these files. Such as XMMS which is available in packages and ports and can play numerous other audio formats.

Apart from the above, there are audio formats which use lossless data compression. Examples are the Free Lossless Audio Codec (FLAC) and TTA. The FLAC implementation has been ported to OpenBSD and may be found under `audio/flac` in packages and ports.

Audio formats using lossy compression (Ogg Vorbis, MP3, WMA, AAC)

Lossy compression methods are often used for audio or other media files. The idea is that an amount of data is thrown away during compression, but in such a way that the compressed result is still very usable and has a good enough quality to be played. The advantage is that these techniques enable much higher compression ratios, resulting in reduced disk space and bandwidth requirements.

A good example is the free, open and unpatented Ogg Vorbis format. To play Ogg Vorbis files, you can use the ogg123 utility, which is bundled in the `audio/vorbis-tools` package. For example:

```
$ ogg123 music.ogg

Audio Device:   Sun audio driver output

Playing: music.ogg
Ogg Vorbis stream: 2 channel, 44100 Hz
Time: 00:02.95
                                02 : 21.45
 of 02:24.40  (133.1 kbps)  Output Buffer  87.5%
```

Of course, Ogg Vorbis plugins exist for many other audio software.

Another example is the very popular MPEG-1 Audio Layer 3 (MP3) encoding, which has, however, its share of licensing and patent issues. Many tools can play MP3 files, just have a look through the `audio` section of the packages and ports system and pick one you like.

How about the proprietary Windows Media Audio (WMA) format? Files of this type can be played using `x11/mplayer` which uses the FFmpeg framework.

A good starting point to learn more about different audio file formats is this Wikipedia article: Audio file formats.

Synthesized sound

MIDI

The Musical Instrument Digital Interface (MIDI) protocol, is handled by MIDI devices. If you don't have a MIDI synthesizer, but you wish to play a standard MIDI file (SMF), you can use software to render MIDI data, generating audio files. By default, the `audio/timidity` port renders MIDI files and play them on the audio device:

```
$ timidity file.mid
```

MOD

A Soundtracker module is a binary format that mixes audio samples with sequencing orders, making it possible to play rather long pieces of digital music with reasonably good quality.

The easiest way to play your favorite MOD files on OpenBSD is probably to use the XMMS software, available through packages and ports. You should install the `-mikmod` subpackage for XMMS to let it use the MikMod sound library, which supports the MOD, S3M, IT and XM module formats.

You will also find a number of so-called "trackers" in the audio section of the packages and ports collection, e.g. tracker, soundtracker. With these trackers you can not only play but also generate your own modules. Note, however, that not every tracker format is supported by the tools in the ports tree. You are always welcome to submit a port of your favorite tracker software.

13.3 How can I play audio CDs in OpenBSD?

It is possible to play audio CDs by either having the CD drive play the disc and send analog audio to the sound card, or by reading the audio data and sending the digital samples to the sound card over the PCI bus.

To play an audio CD using the analog output of your CD-ROM drive, you can

- Use the headphones output, usually at the front side of the drive.

- Connect the audio output at the back side of the drive to your audio card. Yes, this is a supplementary cable next to the data (SATA/IDE/SCSI) and power cables.

A nice command line utility called `cdio(1)`, has been included in the base system. Called without parameters, it will enter interactive mode. If you want to play the CD right away, just enter

```
$ cdio play
```

This will play from the first CD-ROM drive, `cd0`, by default. Note that the user running cdio should have permissions to read the CD-ROM device (e.g. `/dev/rcd0c`). As this device is only readable by root or the operator group by default, for convenience you may want to add the user to the operator group by adjusting this group's line in `/etc/group`. Alternatively, you can modify the file permissions of the device as needed.

Note that you may need to unmute the CD input of the mixer. Just like the outputs, the actual name of this input may vary between systems, but you will be using a command like:

```
$ mixerctl inputs.cd.mute=off
```

It is also possible that there is no analog audio connection between your CD drive and audio device. In this case you could use `cdio`'s `cdplay` command to send the CD audio data to the sound card through the PCI bus.

```
$ cdio cdplay
```

If you prefer a beautiful GUI, there are plenty of X11-based CD players in the packages and ports collection. Just have a look in the `audio` section.

13.4 Can I use OpenBSD to record audio samples?

Yes. Most devices support recording. `aucat(1)` comes with OpenBSD and can be used for recording.

```
$ aucat -o file.wav
```

The above command will start the recording of a file in WAV format. Press

CTRL

-C to finish the recording. The file will contain signed 16-bit stereo samples, sampled at 44.1 kHz. Other sample formats, sample rates and number of channels can be recorded. See the manual for more details.

Use aucat to play the file back:

```
$ aucat -i file.wav
```

If recording seemed to work, but playback of the recording was silent or not what was expected, the mixer probably needs some configuration. Make sure that you select the right device to record from and that the source is unmuted. You can set the necessary parameters using `mixerctl(1)`. For example:

```
inputs.mic.mute=off
inputs.mic.preamp=on
inputs.mic.source=mic0
record.source=mic
record.volume=255,255
record.volume.mute=off
record.mic=255
record.mic.mute=off
```

These are settings for recording from a microphone. Pre-amplifying has been enabled, otherwise the recorded sound can be pretty silent on some systems. However, pre-amplifying can also be quite loud on other systems.

13.5 How do I setup an audio server?

Do I need an audio server?

The `aucat(1)` utility can be used as an audio server, which acts as a layer between the audio(4) driver and audio applications. It aims to:

- Overcome incompatibilities between hardware and applications. For instance the application may not support the encodings or sample rates of the hardware; if so, the server can do the necessary conversions on the fly.
- Allow multiple applications to use the hardware concurrently. For instance, to use a background music player while running an application playing sounds, or to simultaneously use the front speakers for music and the headphone socket for telephony is another example.

If applications you use are compatible with your hardware and you don't plan to run multiple applications concurently, then you don't need an audio server.

How do I setup `aucat(1)`?

There's no configuration file and, in most cases, no tweaking is needed. Typing:

```
$ aucat -l
```

will start the server on the default audio device (the one the `/dev/audio` symlink points to) running at 44.1kHz and using two channels (stereo). This means that applications using stereo at 44.1kHz will run optimally, i.e. without triggering conversion code. If the device doesn't support those parameters, `aucat(1)` will automatically pick another set of parameters.

If you start aucat as root, it will increase its priority, to decrease the probability of buffer underruns or overruns. It can be started at system boot by appending:

```
aucat_flags=""
```

to `/etc/rc.conf.local`.

What latency do I need?

The latency is the time between when a program takes the decision to play a sample and when the user hears the sample. Since audio data is always buffered, this delay is proportional to the audio buffer size. The following values are recommended:

- Real-time synthesizers: 5ms. This is the time it takes between hitting a key on your MIDI keyboard and actually hearing the note. Roughly, 5ms corresponds to the time it takes for the sound to propagate 1.75m.

- Games: 50ms. This is the time between when you see an event and you hear the corresponding sound.

- Movie players and alike: 500ms and more. Such applications "know" the sound to play in advance, and send audio data in such a way that it is played simultaneously with the corresponding picture.

The smaller audio buffers are (to achieve low latency), the larger the probability to overrun/underrun is. Buffer overruns/underruns result in "stuttering" of the sound.

In server mode, `aucat(1)` imposes a minimum latency on all audio applications, and the default latency is around 250ms. If you plan to use applications that require a lower latency, use the "`-b`" option to select the desired latency (expressed in number of frames). For instance, at 44100 samples/second, 50ms latency corresponds to:

44100 samples/second x 0.050 seconds = 2205 samples

then run `aucat(1)` as follows:

```
$ aucat -b 2205 -l
```

Does low latency improve audio-video synchronization?

Synchronizing audio to video doesn't require low latency. Synchronization problems are often caused by the software itself (poor implementation, bugs, ...). Forcing the application to use smaller buffers (by starting aucat(1) in low latency mode) may hide the actual problem in some cases and give the feeling that the software works better, but obviously the right thing to do is to start searching for the corresponding bug.

13.6 What can I do if I have audio problems?

If you do not hear anything when playing audio, it's possible there is a mixer control turned to low or simply muted. See section 13.1 - How do I configure my audio device for configuring the mixer. Please unmute **all** inputs and outputs before reporting a problem.

If sound is distorted, it could be that your sound card only supports a single or limited set of sample rates or encodings. See section 13.1 - How do I configure my audio device for examples of determining what parameters your audio device supports.

If your device only supports unusual encodings or only one or a few sample rates and applications you use do not perform the necessary format conversions, consider using `aucat(1)` as audio server. See section 13.5 - How do I setup an audio server?

If you are still experiencing trouble, here are some things to consider:

- Some old ISA cards have particular quirks:

  Some need to be configured with a different I/O address and IRQ value to avoid conflicts with other hardware. You can easily try different combinations using the User Kernel Configuration (UKC).

  It is possible that a less than optimal driver attaches to the sound device, and that you can get better results using another driver. This is not the easiest thing to spot, but take a closer look at your dmesg(8) output, and find the lines where audio drivers attach. If you see more than one sound driver attaching (or trying to), test them one at a time by disabling some and leaving one enabled using the User Kernel Configuration (UKC).

- Find information about your sound device. Use the documentation, or use an internet search engine to find its specifications. For `ac97(4)` and `azalia(4)` devices, look for documentation for both the controller and the codec. They may actually help you find the source of the problem.

If you believe your device should be working, but for whatever reason isn't, then it's time for a little debugging. The following steps can determine if data is being processed by the DAC.

```
$ cat > /dev/audio < /dev/zero &
```

1

```
 9926
$ audioctl play.{seek,samples,errors}
play.seek=48000
play.samples=3312000
play.errors=0
$ audioctl play.{seek,samples,errors}
play.seek=57600
play.samples=7065600
play.errors=0
$ audioctl play.{seek,samples,errors}
play.seek=48000
play.samples=9379200
play.errors=0
$ kill %1
$ fg %1
cat > /dev/audio < /dev/zero
Terminated
```

Here we see that the processed data count `play.samples` increases each time we check, so data is flowing. We also see that the device is keeping enough data buffered `play.seek` that the device has not underrun any samples `play.errors`. That's good too.

Note that even if you had speakers plugged in when running the above test, you should not have heard anything. The test sends zeros to the device, which is silence for all currently supported default encodings.

Since we know the device can process data, it's a good idea to check the mixer settings again. Make sure all outputs and all inputs are unmuted and are at a reasonable level.

If at this point you are still having problems, it's probably time to file a bug report. Besides the normal bug report information such as a full dmesg and description of the problem, please also include the default output of `mixerctl -v` and the output of the above test for DAC processing.

13.7 How do I use my MIDI instruments?

The Musical Instrument Digital Interface (MIDI) protocol provides a standardized and efficient means to represent musical performance information as electronic data. A MIDI data contain only instructions needed by a synthesizer to play the sounds, rather than the sounds. More information: Tutorial on MIDI and Music Synthesis.

To play MIDI data, a synthesizer connected to a MIDI port of the machine is required. Similarly, to record a MIDI data a MIDI instrument is required (eg.

a MIDI keyboard). Certain sound cards contain embedded MIDI synthesizers that are attached as MIDI ports. Advanced MIDI instruments may contain multiple subparts (synthesizers, keyboards, control surfaces, etc...), they appear as multiple MIDI ports on OpenBSD.

When you already have OpenBSD running, look for MIDI ports in the output of the dmesg(8) command. An example of MIDI ports in a dmesg output is:

```
umidi0 at uhub2 port 2 configuration 1 interface 0 "Roland Roland XV-2020" rev 1.10/1.0
midi0 at umidi0: <USB MIDI I/F>
umidi1 at uhub1 port 2 configuration 1 interface 1 "Evolution Electronics Ltd. USB Keys
midi1 at umidi1: <USB MIDI I/F>
```

It shows three MIDI ports, corresponding to:

- `/dev/rmidi0` - synthesizer connected by USB
- `/dev/rmidi1` - a MIDI master keyboard

These devices are known by `sndio(7)` as `rmidi:0` and `rmidi:1`. To test your MIDI keyboard, you can use the `hexdump(1)` utility to display MIDI data you're playing on it:

```
$ midicat -q rmidi:1 -o - | hexdump -e '1/1 "%02x\n"'
90
3c
71
...
```

The output of the keyboard can be connected to the input of the synthesizer, as follows:

```
$ midicat -q rmidi:0 -q rmidi:1
```

Now you can hear on the synthesizer what you're playing on the MIDI keyboard. Refer to the `midicat(1)` manual page for further information.

The main utility to play standard MIDI files is `midiplay(1)`. Playing a standard MIDI file, in this example through the synthesizer, is as easy as:

```
$ midiplay -f rmidi:0 file.mid
```

To record MIDI files, you can use the `smfrec` utility bundled in the `audio/midish` port, for instance:

```
$ smfrec -d rmidi:0 -i rmidi:1 file.mid
```

will record what is played on the keyboard (`rmidi:1`) while sending it in real-time on the synthesizer (`rmidi:0`) so you can hear what you're playing. More complicated operations like editing, routing, mixing and transforming MIDI data, can be achieved using the rmidish utility bundled in the `audio/midish` port.

13.8 Tell me about Ogg Vorbis and MP3 encoding?

These formats were already mentioned in Playing different kinds of audio. In this section we will give a brief introduction about encoding such files. If you are interested in learning how these audio compression codecs work, further reading may be found through these Wikipedia articles about Vorbis and MP3.

Ogg Vorbis

Encoding raw, WAV or AIFF format audio to Ogg Vorbis may be done with the **oggenc** utility, contained in the `audio/vorbis-tools` package, which is available through OpenBSD's packages and ports system.

Say you have a number of WAV files ready to encode, for example your favorite album you just extracted from its CD. To encode all these files using an approximate bit rate of 192 kbps, you could issue a command like

```
$ oggenc *.wav -b 192
```

When finished, this will give you a set of .ogg files in the current directory. More extensive examples, as well as encoding options, can be found in the oggenc manual page.

MPEG-1 Audio Layer 3 (MP3)

If for some reason you want to use the MP3 format, you can use "Lame ain't an MP3 encoder" (LAME), an educational tool to be used for learning about MP3 encoding. Lame is included in the OpenBSD ports tree. Note that due to MP3 patents, you will not find this package on the official CD sets.

Below is a simple example of encoding a WAV file with a bit rate of 192 kbps:

```
$ lame -b 192 track01.wav track01.mp3
```

For all options and details, please consult the manual page that comes with lame.

13.9 How can I playback video DVDs in OpenBSD?

OpenBSD supports DVD media through the ISO 9660 filesystem which is also used on CD-ROMs, and, since OpenBSD 3.8, also through the newer Universal Disk Format (UDF) filesystem which is found on some DVDs. However, almost all DVD-Video and DVD-ROM discs use the UDF bridge format, which is a combination of the DVD MicroUDF (subset of UDF 1.02) and ISO 9660 filesystems. It is used for backward compatibility reasons.

As most computers with DVD-ROM drives use software decoding, it is recommended to have at least a 350-MHz Pentium II or equivalent CPU to have good quality playback.

Some popular media players, supporting DVD playback, have been ported to OpenBSD. Examples are ogle, mplayer, xine, and kaffeine. Please read the installation instructions that come with these packages, because these tools may need further setup. With these utilities, it is possible to playback the DVD by directly accessing the raw device. Of course, it is also possible to mount a DVD first using `mount_cd9660(8)`, and play the files on this or any other mounted filesystem.

Notes:

- Nearly all DVDs you buy are encrypted using the Content Scrambling System (CSS). To be able to playback these encrypted DVDs, you can use the `libdvd` library, also available through packages and ports.
- Be aware that a region code may be present on your DVD disk(s). This should not be much of a problem when playing DVDs on a computer.

13.10 How do I burn CDs and DVDs?

13.10.1 Introduction and basic setup

You should first make sure your CD/DVD writer has been recognized and configured by the kernel. Most SCSI devices are supported. SATA, IDE/ATAPI and USB devices are supported through SCSI emulation. You will quickly find your device in a `dmesg(8)` output. Just look for lines beginning with "cd", for example

```
cd0 at scsibus0 targ 0 lun 0: <TOSHIBA, CD-ROM XM-5702B, 2826> SCSI0 5/cdrom removable
cd1 at scsibus1 targ 4 lun 0: <PLEXTOR, CD-R PX-R412C, 1.04> SCSI2 5/cdrom removable
```

But `cdrecord -scanbus` does not work!

Yes. OpenBSD uses a different device namespace than the OS for which the cdrecord utility was written. All configured devices should be in the dmesg output, as mentioned above. The information you need is right there.

Error: `mount_cd9660: /dev/cd2c on /mnt/cdrom: No such file or directory`

By default, the OpenBSD installer creates only two cd device nodes, cd0 and cd1. To start using your cd2 device, you must create the necessary device nodes for it. The recommended way to do that is using the `MAKEDEV(8)` (select your specific platform) script:

```
# cd /dev
# ./MAKEDEV cd2
```

In what follows, we will mostly be accessing the CD/DVD writer through the *raw* character device, **not** the *block* device.

Checking CD/DVD writer operation

It is recommended to check whether your CD/DVD writer is working correctly. In this example, I'll be using this USB 2.0 DVD writer:

```
cd2 at scsibus2 targ 1 lun 0: <LITE-ON, DVDRW LDW-851S, GS0C> SCSI0 5/cdrom removable
```

Try to use it by mounting an existing CD/DVD in it. If desired, you could also check the transfer rate you are getting when copying files to your hard disk. The `time(1)` command will be your willing assistant.

If something goes wrong here and you are getting errors during this phase, it is wise to fix the problem and not to start writing a CD/DVD yet.

I want to write a CD here! Can we get on with it?

Before proceeding, it is a good idea to keep a few words of advice in mind:

- Do not run any disk-intensive jobs while writing a CD/DVD. Doing this will reduce the throughput to your CD/DVD writer. If the throughput drops below what the writer is expecting for too long, its buffer will run empty. This phenomenon is also known as a "buffer underrun".
- Prevent shocks during writing as this may cause the laser beam to drift from its track, which may lead to errors on the disc.
- Not every DVD writer supports every DVD format, see below.

13.10.2 Writing CDs

Creating data CD-ROMs

First, you will want to create an ISO 9660 filesystem to put on a CD-ROM. To do this you can use the `mkhybrid(8)` utility in the base system, or the mkisofs utility which comes with the cdrtools package and which does a better job with large file trees. In the examples below, we will use `mkhybrid`, although mkisofs usage is very similar.

As an example usage, let's say I wanted to store the OpenBSD kernel sources in an ISO 9660 image:

```
$ mkhybrid -R -o sys.iso /usr/src/sys

Using ALTQ_RMC.000;1 for  /usr/src/sys/altq/altq_rmclass_debug.h (altq_rmclass.h)
...
Using IEEE8021.00H;1 for  /usr/src/sys/net80211/ieee80211_amrr.c (ieee80211.c)
 10.89% done, estimate finish Sat Nov  3 08:01:23 2007
```

```
 21.78% done, estimate finish Sat Nov  3 08:01:28 2007
...
 87.12% done, estimate finish Sat Nov  3 08:01:31 2007
 98.01% done, estimate finish Sat Nov  3 08:01:32 2007
Total translation table size: 0
Total rockridge attributes bytes: 896209
Total directory bytes: 2586624
Path table size(bytes): 11886
Max brk space used 0
45919 extents written (89 Mb)
```

The -R option tells `mkhybrid` to create Rock Ridge extensions in the ISO 9660 image. The Rock Ridge Interchange Protocol was created to support POSIX filesystem semantics in ISO 9660 filesystems, such as longer file names, ownerships, permissions, file links, soft links, device nodes, deep file hierarchies (more than 8 levels of subdirectories), etc.

If you want the long file names on your CD-ROM to be readable on Windows or DOS systems, you should add the `-J` flag to include Joliet extensions in the ISO 9660 image as well.

After creating the filesystem, you can verify it by mounting the ISO 9660 image. If all is well, you are now ready to burn the CD-R(W). The easiest way to do this is to use the `cdio(1)` utility.

If you are using multi-write media such as CD-RW, you will need to blank the media before burning it.

```
# cdio -f cd1c blank
```

You are now ready to burn the image created in the above example to a blank CD-R(W). You could use a command similar to:

```
# cdio -f cd1c tao sys.iso
```

With the options specified above, we're asking cdio to use the second CD-ROM device as the CD writer.

To verify whether the CD-ROM has been written correctly, you can mount it and check whether everything is there. To mount the filesystem, you should use the **block** device for the CD-ROM drive, which in this case is still the CD writer:

```
# mount /dev/cd1c /mnt/cdrom
```

Creating audio CDs

To burn audio CDs, you can again use `cdio(1)` with the `tao -a` option.

As an example, I'll be making a backup copy of one of my music CDs. This involves two steps:

1. Fetch the audio tracks from the original CD. For example:

```
# cdio -f cd1c cdrip
```

This command will extract a series of WAV files from your second CD-ROM drive to your disk.

2. Write the audio tracks to a blank CD. For example:

```
# cdio -f cd1c tao -a *.wav
```

13.10.3 Writing DVDs

There are a few important things about DVD you should know about before proceeding to write your own DVDs.

Important notes:

- If you really want to know all about DVD, I suggest you read the very extensive DVD FAQ document to start with.
- This section has seen only very limited testing, and we certainly have not tried every possible media and writer combination. Nevertheless, we have had, or have heard of, positive experiences with all of the DVD formats mentioned below. You are welcome to let us know about your successes or failures.

Different DVD formats

There are a number of different DVD formats. Commonly used are the DVD-R, DVD-RW, DVD+R, and DVD+RW formats (R means writable once, RW means it can be rewritten a few thousand times). They are pretty much competing standards.

A pretty different format is DVD-RAM, which was mainly developed as a data drive and has advanced packet writing functions, allowing it to be used like a kind of optical hard disk. DVD-RAM is not recommended for video usage as video gets written to the discs in a format not compatible with normal DVD players.

The important thing is you use media which suit your DVD writer. If you expect compatibility with other DVD players, watch your step and be sure to read this section of the DVD FAQ.

DVD writing speed

It may be useful to point out that DVD speed indications differ from CD-ROM speed indications. The following table gives an overview:

DVD read/write speed	Transfer rate (MB/s)	Equivalent CD-R(W) read/write speed
1x	1.32	9x
2x	2.64	18x
4x	5.28	36x
8c	10.57	72x

As can been seen from the table, the transfer rates are relatively high, and you should check whether your bus (SCSI, IDE/ATAPI, SATA, USB) is performant enough to handle this throughput. Especially the older USB 1.0 and 1.1 interfaces work at slower transfer rates, with maximal rates of 1.5 Mbit/s and 12 Mbit/s, respectively. That means USB 1.0 has a maximal throughput of 178.8 kByte/s and USB 1.1 has a maximal throughput of 1.43 MB/s. USB 2.0 is much faster: 480 Mbit/s or 57.2 MB/s. In general, the speed of SCSI, SATA, and IDE/ATAPI buses should be just fine.

Writing the DVD

Basically, the process is very similar to writing CD-R(W)s. The software used, however, is different. At the moment, the best option is `growisofs` from the `sysutils/dvd+rw-tools` package. This utility writes an ISO 9660 image to the DVD medium. All recordable DVD formats are supported by the dvd+rw-tools.

In case you want to find out more info about the media in your DVD writer (for example if you lost the info label in the jewel case or are just disorganized like me), you can use the `dvd+rw-mediainfo` utility. There are two options to write the DVD:

- Pre-master an ISO 9660 from your data, storing the image on your hard disk; then write this image to the DVD.
- Write an ISO 9660 from your data immediately to the DVD.

I created a pre-mastered ISO 9660 image from the OpenBSD CVS modules (`src`, `xenocara`, `ports` and `www`) contained in the `/cvs` directory on my disk. I used the following command, which looks very similar to the one I used to create the CD-ROM image above.

```
$ mkhybrid -r -o cvs.iso /cvs
```

If desired, check the ISO 9660 filesystem by mounting the image. To write this image (about 2 GB) to an empty DVD disc, one could use:

```
# growisofs -dvd-compat -Z /dev/rcd2c=cvs.iso
Executing 'builtin_dd if=cvs.iso of=/dev/rcd2c obs=32k seek=0'
/dev/rcd2c: pre-formatting blank DVD+RW...
/dev/rcd2c: "Current Write Speed" is 4.1x1385KBps.
  23822336/1545832448 ( 1.5%) @3.9x, remaining 5:19
  42172416/1545832448 ( 2.7%) @3.9x, remaining 5:20
  60522496/1545832448 ( 3.9%) @3.9x, remaining 4:54
...
1504706560/1545832448 (97.3%) @3.9x, remaining 0:07
1523318784/1545832448 (98.5%) @3.9x, remaining 0:04
1541898240/1545832448 (99.7%) @3.9x, remaining 0:00
/dev/rcd2c: flushing cache
/dev/rcd2c: writing lead-out
/dev/rcd2c: reloading tray
```

The -Z option tells growisofs to burn an initial session to the device, which in this case is my DVD writer, attached to cd2. The -dvd-compat option closes the disk, which means no more sessions can be appended to it. This should provide better compatibility with video DVD players and some older DVD-ROM units.

Notice how growisofs indicates the writing speed, in this case 3.9x DVD speed, which is what could be expected from the media and writer combination, as indicated by dvd+rw-mediainfo.

If you are short on disk space and cannot store an ISO 9660 image for a DVD, you can write your data directly to the DVD. Let's first do a dry run, which simulates the creation of the filesystem.

```
# growisofs -dry-run -Z /dev/rcd2c -R /cvs
```

If this succeeds, just leave out the -dry-run option and start burning the DVD.

```
# growisofs -Z /dev/rcd2c -R /cvs
```

It is also possible to append data to an existing DVD, by using the -M option, which merges a new session to an existing one:

```
# growisofs -M /dev/rcd2c -R /mydata
```

For more information about growisofs, refer to the manual page.

When you have finished writing the DVD, mount it and see whether everything you expected to be there, is indeed there.

Why am I not getting the writing speed I expected?

Instead of the above writing output, you may see something like:

```
 4784128/1545832448 ( 0.3%) @0.7x, remaining 26:50
   7929856/1545832448 ( 0.5%) @0.7x, remaining 29:05
  14123008/1545832448 ( 0.9%) @0.7x, remaining 27:06
...
```

which is much slower. It means you are somehow not getting enough throughput on whatever bus your DVD writer is using. In the above example, the USB DVD writer was attached to a machine on which the ehci(4) driver, used by USB 2.0 controllers, failed to initialize properly. As always, you are welcome to provide patches and test results. The DVD writer fell back to the slower USB 1.1 interface, which causes reduced throughput. Indeed, USB 1.1 is limited to 12 Mbit/s, which amounts to 1.43 MB/s or 1.08x in DVD speed terms. The DVD writer falls back to a lower pace than the maximum, to reduce the risk of buffer underruns.

13.11 But I want my media files in format FOO.

Converting between different audio formats

Let's say we want to process the sound recording from chapter 13 - Audio Recording. This recording has been stored in the raw format. It will be useful to convert it, because the raw format does not include headers and the recording parameters will need to be specified at every usage of the file.

One sound conversion tool is `audio/sox`, available through packages and ports. **sox** supports AIFF, AU, MP3, Ogg Vorbis, RIFF WAV and raw formats, as well as some of the more exotic audio formats out there. Below is an example for converting the recording to RIFF WAV format.

```
$ sox -U -c 1 -r 8000 -b myvoice.raw myvoice.wav
```

Note that the specified parameters correspond to the recording parameters specified before the recording. This was just an example. More audio-related libraries and software can be used for audio conversion.

Note: It is not recommended to convert between different lossy compression formats. For instance, the MP3 and Vorbis codecs throw away different parts of an original audio waveform. Therefore, when converting a MP3 file to Ogg Vorbis, the end result will probably sound worse than the original MP3.

Converting between different video formats

It's important to make a clear distinction between

- the container file format - popular examples are MP4, OGG, MPEG, MOV, AVI, ASF.

- the video codec - for example MPEG-1, MPEG-2, MPEG-4 compliant codecs (like Xvid and DivX), FFmpeg, WMV, ... - read this Wikipedia article about video codecs to find out more.

In OpenBSD, support for MPEG and AVI containers is most mature at this time.

Two popular utilities are `multimedia/transcode` and `mencoder` (part of `x11/mplayer`). They use or can use the `libavcodec` library as part of the `graphics/ffmpeg` port, which generates good quality output. You can, of course, also use `ffmpeg` directly. It should also be possible to use the XviD encoder in `multimedia/xvidcore`.

The documentation that comes with these packages, under the form of manual pages or HTML documents in `/usr/local/share/doc`, contains many examples, so it is HIGHLY recommended to read those documents.

13.12 Is it possible to play streaming media under OpenBSD?

Yes, it is. Many audio and video streams will work just fine, on a limited number of platforms. A few of them will not.

This is not meant to be a complete, overly detailed answer to have every possible streaming format work on any hardware architecture. You may want to learn more about streaming media to start with. A slightly dated but still good starting point is this chapter about streaming media from the O'Reilly book titled Designing Web Audio.

The first thing to understand is that there are a number of different streaming protocols around. The streaming **protocol** defines how the streams will be sent over the network. They have been developed to allow efficient transmission of audio/video over the internet in real-time. Mostly, the streaming protocol is a (Layer 7) application protocol, which can use either UDP or TCP (Layer 4) transport protocols. The User Datagram Protocol (UDP) is very suited for this type of application since it doesn't do any retransmission of packets or other overhead. A number of specialized but proprietary protocols have been developed, e.g. Microsoft Media Services (MMS) and the Real Time Streaming Protocol (RTSP). As we will see, HTTP (which uses TCP) is sometimes used as well, even though it does not allow serving streams at a steady bitrate like UDP, RTSP and MMS.

Next, there is the streaming **format**, which is how the audio/video data has been organized and can be played. The most widely used streaming formats are MP3, Real Audio (RA, RM) and Windows Media (ASF), all proprietary technologies. Occasionally you will also encounter streams in the open Ogg Vorbis format.

As an example, I'll explain in a few steps how I get to listen to Radio 1, one of the Belgian national radio stations. Browser-embedded plugins are not available on OpenBSD, so the story is usually not an instant "click and play".

- Determine the streaming protocol and format. This is usually indicated on the website where you access the stream. In this case, I followed the link "Listen live" from the main site, and it's telling me my operating system is not supported. They are being nice by saying I can also listen to their MP3 streams without their embedded Flash player. Apart from that, a list of links to the national radio stations appears, allowing me to proceed to the next step. Note that I used a JavaScript-enabled browser to get this far.

- Figure out the precise URL. Many websites link to a container metafile or playlist (such as M3U, ASX, RAM), which contains the actual location of the stream. Just save the file, and read the URL from it. In my example this is

```
$ ftp http://internetradio.vrt.be/dab/hoeluisteren/pc/help/gebruiksvoorwaarden/stream_11.m3U
```

```
$ cat stream_11.m3U
http://mp3.streampower.be/radio1-mid.mp3
http://mp3.streampower.be/radio1-low.mp3
http://mp3.streampower.be/radio1-high.mp3
```

It looks like I can even choose between low, medium and high quality streams. Other websites may contain some JavaScript code to generate the URL. In that case, the best tip is: dig up the HTML source and scripts it refers to. There is a good chance you can reconstruct the URL from it.

- To play streams, your best bet is probably `x11/mplayer`, which is available through packages and ports. It supports most of the streaming protocols and formats, and has been reported to work on amd64, i386, powerpc and sparc64 platforms. But there are alternatives: `ogg123` from `audio/vorbis-tools` (for Ogg Vorbis streams), `audio/mpg123` and `audio/mpg321` (for MP3 streams), XMMS in `audio/xmms` and the Videolan Client in `x11/vlc`. Continuing the example:

```
$ mplayer http://mp3.streampower.be/radio1-mid.mp3
```

- Optionally, you may want to make it a little easier by including an alias in your `.profile`:

```
alias radio1='mplayer http://mp3.streampower.be/radio1-mid.mp3'
```

Windows Media (ASF) streams will often work, though they may contain data in formats supported only through the `graphics/win32-codecs` port, which runs on i386 only ('pkg_info win32-codecs' will tell you which codecs). Some Real Audio streams can be made to work on i386 using `mplayer` in conjunction with the `graphics/win32-codecs` and `emulators/fedora/base` ports (see this thread on the ports mailing list).

13.13 Can I have Java support in my web browser? (i386 & amd64 only)

The Java plugin is part of the Java Development Toolkit (JDK). For licensing reasons, OpenBSD cannot ship binary packages for the JDK. This means you will have to build it from ports. Further information on building the JDK can be found in chapter 8 - Programming Languages. Once you have finished building the JDK, you can install either the full JDK package or just the Java Runtime Environment (JRE) which is in a subpackage and contains the browser plugin.

Upon installation, instructions are displayed for using the Java plugin with the Firefox or Seamonkey web browser. Create the symlink as explained, and then you should see the Java plugin upon entering "`about:plugins`" in the URL bar.

For KDE's Konqueror web browser, either the java binary must be in your PATH, or its absolute path can be configured from the menu Settings -¿ Configure Konqueror -> Java & JavaScript. By default, the java binary is located in `/usr/local/jre-version/bin/` or `/usr/local/jdk-version/bin/`, depending on whether you installed the JRE or the JDK.

Note: Java support has only been tested with the Firefox, Seamonkey, and Konqueror web browsers. If it works well for you using a different browser, please let us know.

13.14 Can I have Flash support in my web browser?

The Flash plugin is distributed by Adobe in binary form only. Adobe does not provide a native OpenBSD plugin. Considering their security record, we thank them for this neglect.

If you are just looking to watch flash videos from common websites, there are a number of options in packages, including: get_flash_videos, minitube, youtube-dl, get_iplayer and yt. Also, the Gnash project has made a lot of progress lately, and may fill your needs.

Chapter 14

Disk Setup

14.1 Disks and Partitions

The details of setting up disks in OpenBSD vary between platforms, so you should read the installation instructions in the `INSTALL.<arch>` file for your platform to determine the specifics for your system.

Drive identification

OpenBSD handles mass storage with two drivers on most platforms, depending upon the normal command set that kind of device supports:

- `wd(4)`: IDE disks (and devices that look like IDE disks, for example, SATA, MFM or ESDI disks, or a flash device with an appropriate adapter) attached to a `wdc(4)` or `pciide(4)` interface.
- `sd(4)`: Devices that utilize SCSI commands, such as SCSI disks attached to a SCSI adapter, USB disks, SATA disks attached to an `ahci(4)` interface, and disk arrays attached to a RAID controller.

The first drive of a particular type identified by OpenBSD will be drive '`0`', the second will be '`1`', etc. So, the first IDE-like disk will be wd0, the third SCSI-like disk will be `sd2`. If you have two SCSI-like drives and three IDE-like drives on a system, you would have `sd0`, `sd1`, `wd0`, `wd1`, and `wd2` on that machine. The order is based on the order they are found during hardware discovery at boot. There are a few key points to keep in mind:

- Drives may not be numbered in the same order as your boot ROM attempts to boot them (i.e., your system may attempt to boot what OpenBSD identifies as `wd2` or `sd1`). Sometimes you may be able to change this, sometimes not.
- Removing or adding a disk may impact the identity of other drives on the system.

Partitioning

Due to historical reasons, the term "partition" is regularly used for two different things in OpenBSD and this leads to some confusion.

The two types of "partitions" are:

- "disklabel partitions" created with `disklabel(8)` (often called "filesystem partitions").
- "fdisk partitions" created with `fdisk(8)` (often called "partition table partitions" or "Master Boot Record (MBR) partitions").

All OpenBSD platforms use disklabel(8) as the primary way to manage OpenBSD filesystem partitions, but only some platforms also require using

fdisk(8) to manage Partition Table partitions. On the platforms that use fdisk partitions, one fdisk partition is used to hold all of the OpenBSD file systems, this partition is then sliced up into disklabel partitions. These disklabel partitions are labeled "a" through "p". A few of these are "special":

- a – On the boot disk, the 'a' partition is your root partition.
- b – On the boot disk, the 'b' partition is automatically used as a swap partition.
- c – On all disks, the 'c' partition is the entire disk, from the first sector to the last. (Hint: if you wish to totally clear a drive, you write zeros to the 'c' partition of the drive. More commonly, the 'c' partition is used by utilities like 'fdisk' to install boot loaders, partition tables, etc.)

Partition identification

An OpenBSD filesystem is identified by the disk it is on, plus the file system partition on that disk. So, file systems may be identified by identifiers like "sd0a" (the "a" partition of the first "sd" device), "wd2h" (the "h" partition of the third "wd" device), or "sd1c" (the entire second sd device). The device files would be /dev/sd0a for the block device, /dev/rsd0a would be the device file for the "raw" (character) device.

Some utilities will let you use the "shortcut" name of a partition (i.e., "sd0d") or a drive (i.e., "wd1") instead of the actual device name ("/dev/sd0d" or "/dev/wd1c", respectively).

Note again that if you put data on wd2d, then later remove wd1 from the system and reboot, your data is now on wd1d, as your old wd2 is now wd1. However, a drive's identification won't change after boot, so if a USB drive is unplugged or fails, it won't change the identification of other drives until reboot.

Disklabel Unique Identifiers

Disks can also be identified by Disklabel Unique Identifiers (DUIDs), a 16 hex digit number, managed by the diskmap(4) device. This number is generated automatically as a random number when a disklabel is first created, though defaults to all zeros on an existing (pre OpenBSD 4.8) labels. disklabel(8) can be used to change the UID if desired. These UIDs are "persistent" – if you identify your disks this way, drive "f18e359c8fa2522b" will always be f18e359c8fa2522b, no matter what order or how it is attached. You can specify partitions on the disk by appending a period and the partition letter, for example, f18e359c8fa2522b.d is the 'd' partition of the disk f18e359c8fa2522b and will ALWAYS refer to the same chunk of storage, no matter what order the device is attached to the system, or what kind of interface it is attached to.

These UIDs can be used to identify the disks almost anywhere a partition or device would be specified, for example in /etc/fstab or in command lines. Of course, disks and partitions may also be identified in the traditional way,

by device, unit number and partition (i.e., `/dev/sd1f`), and this can be done interchangably.

It is worth noting that the DUID is a property of the disklabel, though as OpenBSD only supports one disklabel per disk, this is mostly academic.

14.2 Using `fdisk(8)`

Be sure to check the `fdisk(8)` man page.

`fdisk(8)` is used on some platforms (i386, amd64, macppc, zaurus and armish) to create a partition recognized by the system boot ROM, into which the OpenBSD disklabel partitions can be placed. Other platforms do not need or use `fdisk(8)`. `fdisk(8)` can also be used for manipulations of the Master Boot Record (MBR), which can impact all operating systems on a computer. Unlike the fdisk-like programs on some other operating systems, OpenBSD's fdisk assumes you know what you want to do, and for the most part, it will let you do what you need to do, making it a powerful tool to have on hand. It will also let you do things you shouldn't or didn't intend to do, so it must be used with care.

Normally, only one OpenBSD fdisk partition will be placed on a disk. That partition will be subdivided by disklabel into OpenBSD filesystem partitions.

To just view your partition table using fdisk, use:

```
# fdisk sd0
```

Which will give an output similar to this:

```
Disk: sd0       geometry: 553/255/63 [8883945 Sectors]
Offset: 0       Signature: 0xAA55
         Starting       Ending       LBA Info:
 #: id    C   H  S -    C   H  S [       start:      size   ]
------------------------------------------------------------------------
*0: A6    3   0  1 -  552 254 63 [       48195:     8835750 ] OpenBSD
 1: 12    0   1  1 -    2 254 63 [          63:       48132 ] Compaq Diag.
 2: 00    0   0  0 -    0   0  0 [           0:           0 ] unused
 3: 00    0   0  0 -    0   0  0 [           0:           0 ] unused
```

In this example we are viewing the fdisk output of the first SCSI-like drive. We can see the OpenBSD partition (`id A6`) and its size. The `*` tells us that the OpenBSD partition is the bootable partition.

In the previous example we just viewed our information. What if we want to edit our partition table? Well, to do so we must use the `-e` flag. This will bring up a command line prompt to interact with fdisk.

```
# fdisk -e wd0
Enter 'help' for information
fdisk: 1> help
```

```
        help            Command help list
        manual          Show entire OpenBSD man page for fdisk
        reinit          Re-initialize loaded MBR (to defaults)
        setpid          Set the identifier of a given table entry
        disk            Edit current drive stats
        edit            Edit given table entry
        flag            Flag given table entry as bootable
        update          Update machine code in loaded MBR
        select          Select extended partition table entry MBR
        swap            Swap two partition entries
        print           Print loaded MBR partition table
        write           Write loaded MBR to disk
        exit            Exit edit of current MBR, without saving changes
        quit            Quit edit of current MBR, saving current changes
        abort           Abort program without saving current changes
fdisk: 1>
```

Here is an overview of the commands you can use when you choose the **-e** flag.

- `help` Display a list of commands that fdisk understands in the interactive edit mode.
- `reinit` Initialize the currently selected, in-memory copy of the boot block. This is a handy way to quickly slap a "full-disk" OpenBSD partition in place, update the boot code, and in general, make the system ready for OpenBSD (and nothing but OpenBSD).
- `disk` Display the current drive geometry that fdisk is using. You are given a chance to edit it if you wish.
- `setpid` Change the partition identifier of the given partition table entry. This command is particularly useful for reassigning an existing partition to OpenBSD.
- `edit` Edit a given table entry in the memory copy of the current boot block. You may edit either in CHS geometry mode, or in sector offsets and sizes.
- `flag` Make the given partition table entry bootable. Only one entry can be marked bootable. If you wish to boot from an extended partition, you will need to mark the partition table entry for the extended partition as bootable.
- `update` Update the machine code in the memory copy of the currently selected boot block.
- `select` Select and load into memory the boot block pointed to by the extended partition table entry in the current boot block.

- `swap` Swaps two MBR entries, so you can re-order the MBR.
- `print` Print the currently selected in-memory copy of the boot block and its MBR table to the terminal.
- `write` Write the in-memory copy of the boot block to disk. You will be asked to confirm this operation.
- `exit` Exit the current level of fdisk, either returning to the previously selected in-memory copy of a boot block, or exiting the program if there is none.
- `quit` Exit the current level of fdisk, either returning to the previously selected in-memory copy of a boot block, or exiting the program if there is none. Unlike exit it does write the modified block out.
- `abort` Quit program without saving current changes.

fdisk tricks and tips

- `fdisk(8)` offers the ability to specify partitions both in raw sectors and in Cylinder/Head/Sector formats. Both options are given for a reason – some tasks are easier accomplished one way, others the other way. Don't lock yourself into only using one option.
- A totally blank disk will need to have the master boot record's boot code written to the disk before it can boot. You can use the "reinit" or "update" options to do this. If you fail to do this, you can write a valid partition table with fdisk, but not have a bootable disk. You may wish to update the existing boot code anyway if you are uncertain of its origin.
- If your system has a "maintenance" or "diagnostic" partition, it is recommended that you leave it in place or install it BEFORE installing OpenBSD.
- For historical reasons, "q" saves changes and exits the program, and "x" exits without saving. This is the opposite of what many people are now used to in other environments. fdisk(8) does not warn before saving the changes, so use with care.

14.3 Using OpenBSD's `disklabel(8)`

What is `disklabel(8)`?

First, be sure to read the `disklabel(8)` man page.

The details of setting up disks in OpenBSD varies somewhat between platforms. For i386, amd64, macppc, zaurus, and armish, disk setup is done in two stages. First, the OpenBSD slice of the hard disk is defined using `fdisk(8)`, then that slice is subdivided into OpenBSD partitions using `disklabel(8)`.

All OpenBSD platforms, however, use `disklabel(8)` as the primary way to manage OpenBSD partitions. Platforms that also use `fdisk(8)` place all the `disklabel(8)` partitions in a single fdisk partition.

Labels hold certain information about your disk, like your drive geometry and information about the filesystems on the disk. The disklabel is then used by the bootstrap program to access the drive and to know where filesystems are contained on the drive. You can read more in-depth information about disklabel in the `disklabel(5)` man page.

On some platforms, disklabel helps overcome architecture limitations on disk partitioning. For example, on i386, you can have 4 primary partitions, but with `disklabel(8)`, you use one of these 'primary' partitions to store all of your OpenBSD partitions (for example, '`swap`', '`/`', '`/usr`', '`/var`', etc.), and you still have 3 more partitions available for other OSs.

`disklabel(8)` during OpenBSD's install

One of the major parts of OpenBSD's install is your initial creation of labels. During the install you use disklabel(8) to create your separate partitions. As part of the install process, you can define your mount points from within disklabel(8), but you can change these later in the install or post-install, as well.

There is not one "right" way to label a disk, but there are many wrong ways. Before attempting to label your disk, see this discussion on partitioning and partition sizing.

For an example of using `disklabel(8)` during install, see the Setting up disks part of the Installation Guide.

Using `disklabel(8)` after install

After install, one of the most common reasons to use `disklabel(8)` is to look at how your disk is laid out. The following command will show you the current disklabel, without modifying it:

```
# disklabel wd0 <-- Or whatever disk device you'd like to view
type: ESDI
disk: ESDI/IDE disk
label: SAMSUNG HD154UI
uid: d920a43a5a56ad5f
flags:
bytes/sector: 512
sectors/track: 63
tracks/cylinder: 16
sectors/cylinder: 1008
cylinders: 2907021
total sectors: 2930277168
boundstart: 64
boundend: 2930272065
```

```
drivedata: 0

16 partitions:
#                size           offset  fstype [fsize bsize  cpg]
  a:          1024064               64  4.2BSD   2048 16384    1 # /
  b:          4195296          1024128    swap
  c:       2930277168                0  unused
  d:          4195296          5219424  4.2BSD   2048 16384    1 # /usr
  e:          4195296          9414720  4.2BSD   2048 16384    1 # /tmp
  f:         20972448         13610016  4.2BSD   2048 16384    1 # /var
  h:          2097632         34582464  4.2BSD   2048 16384    1 # /home
```

Note how this disk has only part of its disk space allocated at this time. Disklabel offers two different modes for editing the disklabel, a built-in command-driven editor (this is how you installed OpenBSD originally), and a full editor, such as `vi(1)`. You may find the command-driven editor "easier", as it guides you through all the steps and provides help upon request, but the full-screen editor has definite use, too.

Let's add a partition to the above system.

Warning: Any time you are fiddling with your disklabel, you are putting all the data on your disk at risk. Make sure your data is backed up before editing an existing disklabel!

We will use the built-in command-driven editor, which is invoked using the "`-E`" option to `disklabel(8)`.

```
# disklabel -E wd0
...
> a k
offset: [36680096]
size: [2893591969] 1T
Rounding to cylinder: 2147483536
FS type: [4.2BSD]
> p m
OpenBSD area: 64-2930272065; size: 1430796.9M; free: 364310.8M
#                size           offset  fstype [fsize bsize  cpg]
  a:           500.0M               64  4.2BSD   2048 16384    1 # /
  b:          2048.5M          1024128    swap
  c:       1430799.4M                0  unused
  d:          2048.5M          5219424  4.2BSD   2048 16384    1 # /usr
  e:          2048.5M          9414720  4.2BSD   2048 16384    1 # /tmp
  f:         10240.5M         13610016  4.2BSD   2048 16384    1 # /var
  h:          1024.2M         34582464  4.2BSD   2048 16384    1 # /home
  k:       1048575.9M         36680192  4.2BSD   8192 65536    1
> q
Write new label?: [y]
```

In this case, `disklabel(8)` was kind enough to calculate a good starting offset for the partition. In many cases, it will be able to do this, but if you have "holes" in the disklabel (i.e., you deleted a partition, or you just like making your life miserable) you may need to sit down with a paper and pencil to calculate the proper offset. Note that while `disklabel(8)` does some sanity checking, it is very possible to do things very wrong here. Be careful, understand the meaning of the numbers you are entering.

On most OpenBSD platforms, there are sixteen disklabel partitions available, labeled "a" through "p". (some "specialty" systems may have only eight). Every disklabel should have a 'c' partition, with an "fstype" of "unused" that covers the entire physical drive. If your disklabel is not like this, it must be fixed, the "D" option (below) can help. Never try to use the "c" partition for anything other than accessing the raw sectors of the disk, do not attempt to create a file system on "c". On the boot device, "a" is reserved for the root partition, and "b" is the swap partition, but only the boot device makes these distinctions. Other devices may use all fifteen partitions other than "c" for file systems.

Disklabel tricks and tips

- **Get help**: In the command-driven mode, hitting "?" will produce a list of available commands. "M" will show the man page for `disklabel(8)`.

- **Reset to default**: In some cases, you may wish to completely restart from scratch and delete all existing disklabel information. The "D" command will reset the label back to default, as if there had never been a disklabel on the drive.

- **Duplicating a disklabel**: In some cases, you may wish to duplicate the partitioning from one disk to another, but not precisely (for example, you wish to have the same partitions, but on different sizes of drives). Use the '-e' (full-screen editor) mode of disklabel(8) to capture the partitions of the "model" drive, paste it into the new drive, remove the model's 'c' partition, save, and you have copied the disk layout to the other drive without altering its basic parameters.

- (sparc/sparc64) **Don't put swap at the very beginning of your disk**. While Solaris often puts swap at the very beginning of a disk, OpenBSD requires the boot partition to be at the beginning of the disk.

- (fdisk platforms) **Leave first track free**: On platforms using fdisk(8), you should leave the first logical track unused, both in disklabel(8) and in fdisk(8). On "modern" computers (i.e., almost everything that will run OpenBSD), the exact amount doesn't really matter, though for performance reasons on the newest disks, having partitions aligned at 4k boundaries is good for performance. For this reason, OpenBSD now defaults to starting the first partition at block 64 instead of 63.

- **Devices without a disklabel**: If a device does not currently have an OpenBSD disklabel on it but has another file system (for example, a disk with a pre-existing FAT32 file system), the OpenBSD kernel will "create" one in memory, and that can form the basis of a formal OpenBSD disklabel to be stored on disk. However, if a disklabel is created and saved to disk, and a non-OpenBSD file system is added later, the disklabel will not be automatically updated. You must do this yourself if you wish OpenBSD to be able to access this file system. More on this below.
- **"q" vs. "x"**: For historical reasons, while in the command-driven editor mode, "q" saves changes and exits the program, and "x" exits without saving. This is the opposite of what many people are now used to in other environments. disklabel(8) does warn before saving the changes, though it will "x" quickly and quietly.
- **Auto-partitioning**: New users are encouraged to use the 'A' command to auto-create a recommended disklabel. You can then edit or alter the auto-created label as you need.

14.4 Adding extra disks in OpenBSD

Well once you get your disk installed **PROPERLY** you need to use `fdisk(8)` (*i386 only*) and `disklabel(8)` to set up your disk in OpenBSD.

For i386 folks, start with fdisk. Other architectures can ignore this. In the below example we're adding a third SCSI-like drive to the system.

```
# fdisk -i sd2
```

This will initialize the disk's "real" partition table for exclusive use by OpenBSD. Next you need to create a disklabel for it. This will seem confusing.

```
# disklabel -e sd2

(screen goes blank, your $EDITOR comes up)
type: SCSI
...bla...
sectors/track: 63
total sectors: 6185088
...bla...
16 partitions:
#        size   offset    fstype   [fsize bsize   cpg]
  c:  6185088        0    unused        0     0         # (Cyl.    0 - 6135)
  d:  1405080       63    4.2BSD     1024  8192    16   # (Cyl.    0*- 1393*)
  e:  4779945  1405143    4.2BSD     1024  8192    16   # (Cyl. 1393*- 6135)
```

First, ignore the 'c' partition, it's always there and is for programs like disklabel to function! Fstype for OpenBSD is 4.2BSD. Total sectors is the

total size of the disk. Say this is a 3 gigabyte disk. Three gigabytes in disk manufacturer terms is 3000 megabytes. So divide 6185088/3000 (use `bc(1)`). You get 2061. So, to make up partition sizes for a, d, e, f, g, ... just multiply X*2061 to get X megabytes of space on that partition. The offset for your first new partition should be the same as the "sectors/track" reported earlier in disklabel's output. For us it is 63. The offset for each partition afterwards should be a combination of the size of each partition and the offset of each partition (except the 'c' partition, since it has no play into this equation.)

Or, if you just want one partition on the disk, say you will use the whole thing for web storage or a home directory or something, just take the total size of the disk and subtract the sectors per track from it. 6185088-63 = 6185025. Your partition is

```
d:  6185025       63    4.2BSD     1024  8192    16
```

If all this seems needlessly complex, you can just use disklabel -E to get the same partitioning mode that you got on your install disk! There, you can just use "96M" to specify "96 megabytes", or 96G for 96 gigs.

That was a lot. But you are not finished. Finally, you need to create the filesystem on that disk using `newfs(8)`.

```
# newfs sd2d
```

Or whatever your disk was named as per OpenBSD's disk numbering scheme. (Look at the output from `dmesg(8)` to see what your disk was named by OpenBSD.)

Now figure out where you are going to mount this new partition you just created. Say you want to put it on `/u`. First, make the directory `/u`. Then, mount it.

```
# mount /dev/sd2d /u
```

Finally, add it to `/etc/fstab(5)`.

```
/dev/sd2d /u ffs rw 1 1
```

What if you need to migrate an existing directory like `/usr/local`? You should mount the new drive in `/mnt` and use

```
cpio -pdum
```

to copy /usr/local to the `/mnt` directory. Edit the `/etc/fstab(5)` file to show that the `/usr/local` partition is now `/dev/sd2d` (your freshly formatted partition). Example:

```
/dev/sd2d /usr/local ffs rw 1 1
```

Reboot into single user mode with `boot -s`, move the existing `/usr/local` to `/usr/local-backup` (or delete it if you feel lucky) and create an empty directory `/usr/local`. Then reboot the system, and voila, the files are there!

14.5 How is swap handled?

14.5.1 About swap

Historically, all kinds of rules have been tossed about to guide administrators on how much swap to configure on their machines. The problem, of course, is there are few "normal" application.

One non-obvious use for swap is to be a place the kernel can dump a copy of what is in core in the event of a system panic for later analysis. For this to work, you must have a swap partition (not a swap file) at least as large as your RAM. By default, the system will save a copy of this dump to `/var/crash` on reboot, so if you wish to be able to do this automatically, you will need sufficient free space on `/var`. However, you can also bring the system up single-user, and use `savecore(8)` to dump it elsewhere.

Many types of systems may be appropriately configured with no swap at all. For example, firewalls should not swap in normal operation. Machines with flash storage generally should not swap. If your firewall is flash based, you may benefit (slightly) by not allocating a swap partition, though in most other cases, a swap partition won't actually hurt anything; most disks have more than enough space to allocate a little to swap.

There are all kinds of tips about optimizing swap (where on the disk, separate disks, etc.), but if you find yourself in a situation where optimizing swap is an issue, you probably need more RAM. In general, the best optimization for swap is to not need it.

In OpenBSD, swap is managed with the `swapctl(8)` program, which adds, removes, lists and prioritizes swap devices and files.

14.5.2 Swapping to a partition

On OpenBSD, the '`b`' partition of the boot drive is used by default and automatically for swap. No configuration is needed for this to take place. If you do not wish to use swap on the boot disk, do not define a "`b`" partition. If you wish to use swap on other partitions or on other disks, you need to define these partitions in `/etc/fstab` with lines something like:

```
/dev/sd3b none swap sw 0 0
/dev/sd3d none swap sw 0 0
```

14.5.3 Swapping to a file

(Note: if you are looking to swap to a file because you are getting "virtual memory exhausted" errors, you should try raising the per-process limits first with `csh(1)`'s `unlimit(1)`, or `sh(1)`'s `ulimit(1)`.)

Sometimes, your initial guess about how much swap you need proves to be wrong, and you have to add additional swap space, occasionally in a hurry (as in, "Geez, at the rate it is burning swap, we'll be wedged in five minutes"). If

you find yourself in this position, adding swap space as a file on an existing file system can be a quick fix.

The file must not reside on a filesystem which has SoftUpdates enabled (they are disabled by default). To start out, you can see how much swap you currently have and how much you are using with the `swapctl(8)` utility. You can do this by using the command:

```
$ swapctl -l
Device      512-blocks     Used    Avail Capacity  Priority
swap_device      65520        8    65512     0%    0
```

This shows the devices currently being used for swapping and their current statistics. In the above example there is only one device named "`swap_device`". This is the predefined area on disk that is used for swapping. (Shows up as partition b when viewing disklabels) As you can also see in the above example, that device isn't getting much use at the moment, but for the purposes of this document, we will act as if an extra 32M is needed.

The first step to setting up a file as a swap device is to create the file. It's best to do this with the `dd(1)` utility. Here is an example of creating the file `/var/swap` that is 32M in size.

```
$ sudo dd if=/dev/zero of=/var/swap bs=1k count=32768
32768+0 records in
32768+0 records out
33554432 bytes transferred in 20 secs (1677721 bytes/sec)
```

Once this has been done, we can turn on swapping to that device. Use the following command to turn on swapping to this device

```
$ sudo chmod 600 /var/swap
$ sudo swapctl -a /var/swap
```

Now we need to check to see if it has been correctly added to the list of our swap devices.

```
$ swapctl -l
Device      512-blocks     Used    Avail Capacity  Priority
swap_device      65520        8    65512     0%    0
/var/swap        65536        0    65536     0%    0
Total           131056        8   131048     0%
```

Now that the file is setup and swapping is being done, you need to add a line to your `/etc/fstab` file so that this file is configured on the next boot time also. If this line is not added, your won't have this swap device configured.

```
$ cat /etc/fstab
/dev/wd0a / ffs rw 1 1
/var/swap /var/swap swap sw 0 0
```

14.6 Soft Updates

Soft Updates is based on an idea proposed by Greg Ganger and Yale Patt and developed for FreeBSD by Kirk McKusick. SoftUpdates imposes a partial ordering on the buffer cache operations which permits the requirement for synchronous writing of directory entries to be removed from the FFS code. Thus, a large performance increase is seen in disk writing performance.

Enabling soft updates must be done with a mount-time option. When mounting a partition with the `mount(8)` utility, you can specify that you wish to have soft updates enabled on that partition. Below is a sample `/etc/fstab(5)` entry that has one partition `sd0a` that we wish to have mounted with soft updates.

```
/dev/sd0a / ffs rw,softdep 1 1
```

Note to sparc users: Do not enable soft updates on sun4 or sun4c machines. These architectures support only a very limited amount of kernel memory and cannot use this feature. However, sun4m machines are fine.

14.7 How do OpenBSD/i386 and OpenBSD/amd64 boot?

The boot process for OpenBSD/i386 and OpenBSD/amd64 is not trivial, and understanding how it works can be useful to troubleshoot a problem when things don't work. There are four key pieces to the boot process:

1. ***Master Boot Record (MBR)***: The Master Boot Record is the first 512 bytes on the disk. It contains the primary partition table and a small program to load the Partition Boot Record (PBR). Note that in some environments, the term "MBR" is used to refer to only the code portion of this first block on the disk, rather than the whole first block (including the partition table). It is critical to understand the meaning of "initialize the MBR" – in the terminology of OpenBSD, it would involve rewriting the entire MBR, clearing any existing partition table, not just the code, as it might on some systems. You will often not want to do this. Instead, use `fdisk(8)`'s "`-u`" command line option ("`fdisk -u wd0`") to (re)install the MBR boot code.

   While OpenBSD includes its own MBR code, you are not obliged to use it, as virtually any MBR code can boot OpenBSD. The MBR is manipulated by the fdisk(8) program, which is used both to edit the partition table, and also to install the MBR code on the disk.

   OpenBSD's MBR announces itself with the message:

   ```
   Using drive 0, partition 3.
   ```

showing the disk and partition it is about to load the PBR from. In addition to the obvious, it also shows a trailing period ("`.`"), which indicates this machine is capable of using LBA translation to boot. If the machine were incapable of using LBA translation, the above period would have have been replaced with a semicolon ("`;`"), indicating CHS translation:

```
Using drive 0, partition 3;
```

Note that the trailing period or semicolon can be used as an indicator of the "new" OpenBSD MBR, introduced with OpenBSD 3.5.

2. ***Partition Boot Record (PBR)***: The Partition Boot Record, also called the PBR or `biosboot(8)` (after the name of the file that holds the code) is the first 512 bytes of the OpenBSD partition of the disk. The PBR is the "first-stage boot loader" for OpenBSD. It is loaded by the MBR code, and has the task of loading the OpenBSD second-stage boot loader, `boot(8)`. Like the MBR, the PBR is a very tiny section of code and data, only 512 bytes, total. That's not enough to have a fully filesystem-aware application, so rather than having the PBR locate `/boot` on the disk, the BIOS-accessible location of `/boot` is physically coded into the PBR at installation time.

   The PBR is installed by `installboot(8)`, which is further described later in this document. The PBR announces itself with the message:

   ```
   Loading...
   ```

   printing a dot for every file system block it attempts to load. Again, the PBR shows if it is using LBA or CHS to load, if it has to use CHS translation, it displays a message with a semicolon:

   ```
   Loading;...
   ```

3. ***Second Stage Boot Loader, /boot***: `/boot` is loaded by the PBR, and has the task of accessing the OpenBSD file system through the machine's BIOS, and locating and loading the actual kernel. boot(8) also passes various options and information to the kernel.

   `boot(8)` is an interactive program. After it loads, it attempts to locate and read `/etc/boot.conf`, if it exists (which it does not on a default install), and processes any commands in it. Unless instructed otherwise by `/etc/boot.conf`, it then gives the user a prompt:

   ```
   probing: pc0 com0 com1 apm mem[636k 190M a20=on]
   disk: fd0 hd0+
   >> OpenBSD/i386 BOOT 3.17
   boot>
   ```

It gives the user (by default) five seconds to start giving it other tasks, but if none are given before the timeout, it starts its default behavior: loading the kernel, bsd, from the root partition of the first hard drive. The second-stage boot loader probes (examines) your system hardware, through the BIOS (as the OpenBSD kernel is not loaded). Above, you can see a few things it looked for and found:

- `pc0` - the standard keyboard and video display of an i386 system.
- `com0`, `com1` - Two serial ports
- `apm` - Advanced Power Management BIOS functions
- `636k 190M` - The amount of conventional (below 1M) and extended (above 1M) memory it found
- `fd0 hd0+` - The BIOS disk devices found, in this case, one floppy and one hard disk.

The '+' character after the "`hd0`" indicates that the BIOS has told `/boot` that this disk can be accessed via LBA. When doing a first-time install, you will sometimes see a '*' after a hard disk – this indicates a disk that does not seem to have a valid OpenBSD disk label on it.

4. **Kernel**: **`/bsd`**: This is the goal of the boot process, to have the OpenBSD kernel loaded into RAM and properly running. Once the kernel has loaded, OpenBSD accesses the hardware directly, no longer through the BIOS.

So, the very start of the boot process could look like this:

```
Using drive 0, partition 3.                        <- MBR
Loading....                                        <- PBR
probing: pc0 com0 com1 apm mem[636k 190M a20=on] <- /boot
disk: fd0 hd0+
>> OpenBSD/i386 BOOT 3.17
boot>
booting hd0a:/bsd 4464500+838332 [58+204240+181750]=0x56cfd0
entry point at 0x100120

[ using 386464 bytes of bsd ELF symbol table ]
Copyright (c) 1982, 1986, 1989, 1991, 1993       <- Kernel
        The Regents of the University of California.  All rights reserved.
Copyright (c) 1995-2008 OpenBSD.  All rights reserved.  http://www.OpenBSD.org

OpenBSD 5.0 (GENERIC) #43: Wed Aug 17 10:10:52 MDT 2011
    deraadt@i386.openbsd.org:/usr/src/sys/arch/i386/compile/GENERIC
   ...
```

What can go wrong

- **Bad/invalid/incompatible MBR**: Usually, a used hard disk has some MBR code in place, but if the disk is new or moved from a different platform, AND you don't answer "w" to the "`Use (W)hole disk or (E)dit the MBR?`" question of the installation process, you may end up with a disk without a valid MBR, and thus, it will not be bootable, even though it has a valid partition table.

  You may install the OpenBSD MBR on your hard disk using the fdisk program. Boot from your install media, choose "`Shell`" to get a command prompt:

  ```
  # fdisk -u wd0
  ```

  You may also install a specific MBR to disk using fdisk:

  ```
  # fdisk -u -f /usr/mdec/mbr wd0
  ```

  which will install the file `/usr/mdec/mbr` as your system's MBR. This particular file on a standard OpenBSD install happens to be the standard MBR that is also built into fdisk, but any other MBR could be specified here.

- **Invalid `/boot` location installed in PBR**: When installboot(8) installs the partition boot record, it writes the block number and offset of `/boot`'s inode into the PBR. Therefore, deleting and replacing `/boot` without re-running `installboot(8)` will render your system unbootable, as the PBR will load whatever happens to be pointed to by the inode specified in it, which will almost certainly no longer be the desired second-stage boot loader! Since `/boot` is being read using BIOS calls, old versions of the PBR were sensitive to BIOS disk translation. If you altered the drive's geometry (i.e., took it out of one computer that uses CHS translation and moving it into one that uses LBA translation, or even changed a translation option in your BIOS), it would have appeared to the BIOS to be in a different location (a different numerical block must be accessed to get the same data from the disk), so you would have had to run `installboot(8)` before the system could be rebooted. The new (as of OpenBSD 3.5 and later) PBR is much more tolerant to changes in translation.

As the PBR is very small, its range of error messages is pretty limited, and somewhat cryptic. Most likely messages are:

- **ERR R** – BIOS returned an error when trying to read a block from the disk. Usually means exactly what it says: your disk wasn't readable.

- **ERR M** – An invalid `magic(5)` number was read in the second-stage bootloader's header. This generally means whatever it was that was read in was NOT `/boot`, usually meaning `installboot(8)` was run incorrectly, the `/boot` file was altered, or you have exceeded your BIOS's ability to read a large disk.

Other error messages are detailed in the `biosboot(8)` manual page.
For more information on the i386 boot process, see:

- `boot_i386(8)`
- http://www.ata-atapi.com/hiw.html Hale Landis' "How it Works" documents.

14.8 What are the issues regarding large drives with OpenBSD?

OpenBSD supports both FFS and FFS2 (also known as UFS and UFS2) file systems. FFS is the historic OpenBSD file system, FFS2 is new as of 4.3. Before looking at the limits of each system, we need to look at some more general system limits.

Of course, the ability of file system and the abilities of particular hardware are two different things. A newer 250G IDE hard disk may have issues on older (pre ¿137G standards) interfaces (though for the most part, they work just fine), and some very old SCSI adapters have been seen to have problems with more modern drives, and some older BIOSs will hang when they encounter a modern sized hard disk. You must respect the abilities of your hardware and boot code, of course.

Partition size and location limitations

Unfortunately, the full ability of the OS isn't available until AFTER the OS has been loaded into memory. The boot process has to utilize (and is thus limited by) the system's boot ROM.

For this reason, the entire `/bsd` file (the kernel) must be located on the disk within the boot ROM addressable area. This means that on some older i386 systems, the root partition must be completely within the first 504M, but newer computers may have limits of 2G, 8G, 32G, 128G or more. It is worth noting that many relatively new computers which support larger than 128G drives actually have BIOS limitations of booting only from within the first 128G. You can use these systems with large drives, but your root partition must be within the space supported by the boot ROM.

Note that it is possible to install a 40G drive on an old 486 and load OpenBSD on it as one huge partition, and think you have successfully violated the above rule. However, it might come back to haunt you in a most unpleasant way:

- You install on the 40G / partition. It works, because the base OS and all its files (including `/bsd`) are within the first 504M.
- You use the system, and end up with more than 504M of files on it.
- You upgrade, build your own kernel, whatever, and copy your new /bsd over the old one.
- You reboot.
- You get a message such as "ERR M" or other problems on boot.

Why? Because when you copied "over" the new /bsd file, it didn't overwrite the old one, it got relocated to a new location on the disk, probably outside the 504M range the BIOS supported. The boot loader was unable to fetch the file `/bsd`, and the system hung.

To get OpenBSD to boot, the boot loaders (biosboot(8) and /boot in the case of i386/amd64) and the kernel (`/bsd`) must be within the boot ROM's supported range, and within their own abilities. To play it safe, the rule is simple:

The entire root partition must be within the computer's BIOS (or boot ROM) addressable space.

Some non-i386 users think they are immune to this, however most platforms have some kind of boot ROM limitation on disk size. Finding out for sure what the limit is, however, can be difficult.

This is another good reason to partition your hard disk, rather than using one large partition.

`fsck(8)` time and memory requirements

Another consideration with large file systems is the time and memory required to `fsck(8)` the file system after a crash or power interruption. One should not put a 120G file system on a system with 32M of RAM and expect it to successfully `fsck(8)` after a crash. A rough guideline is the system should have at least 1M of available memory for every 1G of disk space to successfully fsck the disk. Swap can be used here, but at a very significant performance penalty, so severe that it is usually unacceptable, except in special cases.

The time required to fsck the drive may become a problem as the file system size expands, but you only have to fsck the disk space that is actually allocated to mounted filesystems. This is another reason NOT to allocate all your disk space Just Because It Is There. Keeping file systems mounted RO or not mounted helps keep them from needing to be fsck(8)ed after tripping over the power cord.

Don't forget that if you have multiple disks on the system, they could all end up being `fsck(8)`ed after a crash at the same time, so they could require more RAM than a single disk.

By the time one gets to somewhat larger than 1TB file system with default fragment and block sizes, fsck will require 1GB RAM to run, which is the

application limit under OpenBSD. Larger fragments and/or blocks will reduce the number of inodes, and allow for larger file systems.

FFS vs. FFS2

Using FFS, OpenBSD supports an individual file system of up to 231-1, or 2,147,483,647 blocks, and as each block is 512 bytes, that's a tiny amount less than 1T. FFS2 is capable of much larger file systems, though other limits will be reached long before the file system limits will be reached.

The boot/installation kernels *only support FFS*, not FFS2, so key system partitions (`/`, `/usr`, `/var`, `/tmp`) should not be FFS2, or severe maintenance problems can arise (there should be no reason for those partitions to be that large, anyway). For this reason, very large partitions should only be used for "non-system" partitions, for example, `/home`, `/var/www/`, `/bigarray`, etc.

Note that not all controllers and drivers support large disks. For example, `ami(4)` has a limit of 2TB per logical volume. Always be aware of what was available when a controler or interface was manufactured, and don't just rely on "the connectors fit".

14.9 Installing Bootblocks - i386/amd64 specific

OpenBSD has a very robust boot loader that is quite indifferent to drive geometries, however, it is sensitive to where the file `/boot` resides on the disk. If you do something that causes `boot(8)` to be moved to a new place on the disk (actually, a new inode), you will "break" your system, preventing it from booting properly. To fix your boot block so that you can boot normally, just put a boot CDROM in your drive (or use a boot floppy) and at the boot prompt, type "boot hd0a:/bsd" to force it to boot from the first hard disk (and not the CD or floppy). Your machine should come up normally. You now need to reinstall the first-stage boot loader (`biosboot(8)`) based on the position of the `/boot` file, using the `installboot(8)` program.

Our example will assume your boot disk is sd0 (but for IDE it would be wd0, etc.):

```
# cd /usr/mdec; ./installboot /boot biosboot sd0
```

Note that "`/boot`" is the physical location of the file "`boot`" you wish to use when the system boots normally as the system is currently mounted. If your situation were a little different and you had booted from the CD and mounted your 'a' partition on `/mnt`, this would probably be "`/mnt/boot`" instead. `installboot(8)` does two things here – it installs the file "`biosboot`" to where it needs to be in the Partition Boot Record, and modifies it with the physical location of the "`/boot`" file.

14.10 Preparing for disaster: Backing up and Restoring from tape

Introduction:

If you plan on running what might be called a production server, it is advisable to have some form of backup in the event one of your fixed disk drives fails, or the data is otherwise lost.

This information will assist you in using the standard `dump(8)`/`restore(8)` utilities provided with OpenBSD. More advanced backup utilities, such as "Amanda" and "Bacula" are available through packages for backing up multiple servers to disk and tape.

Backing up to tape:

Backing up to tape requires knowledge of where your file systems are mounted. You can determine how your filesystems are mounted using the `mount(8)` command at your shell prompt. You should get output similar to this:

```
# mount
/dev/sd0a on / type ffs (local)
/dev/sd0h on /usr type ffs (local)
```

In this example, the root (`/`) filesystem resides physically on sd0a which indicates a SCSI-like fixed disk 0, partition a. The `/usr` filesystem resides on `sd0h`, which indicates SCSI-like fixed disk 0, partition h.

Another example of a more advanced mount table might be:

```
# mount
/dev/sd0a on / type ffs (local)
/dev/sd0d on /var type ffs (local)
/dev/sd0e on /home type ffs (local)
/dev/sd0h on /usr type ffs (local)
```

In this more advanced example, the root (`/`) filesystem resides physically on `sd0a`. The `/var` filesystem resides on `sd0d`, the `/home` filesystem on `sd0e` and finally `/usr` on `sd0h`.

To backup your machine you will need to feed dump the name of each fixed disk partition. Here is an example of the commands needed to backup the simpler mount table listed above:

```
# /sbin/dump -0au -f /dev/nrst0 /dev/rsd0a
# /sbin/dump -0au -f /dev/nrst0 /dev/rsd0h
# mt -f /dev/rst0 rewind
```

For the more advanced mount table example, you would use something similar to:

```
# /sbin/dump -0au -f /dev/nrst0 /dev/rsd0a
# /sbin/dump -0au -f /dev/nrst0 /dev/rsd0d
# /sbin/dump -0au -f /dev/nrst0 /dev/rsd0e
# /sbin/dump -0au -f /dev/nrst0 /dev/rsd0h
# mt -f /dev/rst0 rewind
```

You can review the `dump(8)` man page to learn exactly what each command line switch does. Here is a brief description of the parameters used above:

- `0` - Perform a level 0 dump, get everything
- `a` - Attempt to automatically determine tape media length
- `u` - Update the file /etc/dumpdates to indicate when backup was last performed
- `f` - Which tape device to use (/dev/nrst0 in this case)

Finally which partition to backup (`/dev/rsd0a`, etc.)

The `mt(1)` command is used at the end to rewind the drive. Review the mt man page for more options (such as eject).

If you are unsure of your tape device name, use `dmesg` to locate it. An example tape drive entry in dmesg might appear similar to:

```
st0 at scsibus0 targ 5 lun 0: <ARCHIVE, Python 28388-XXX, 5.28>
```

You may have noticed that when backing up, the tape drive is accessed as device name "`nrst0`" instead of the "`st0`" name that is seen in `dmesg`. When you access `st0` as `nrst0` you are accessing the same physical tape drive but telling the drive to not rewind at the end of the job and access the device in raw mode. To back up multiple file systems to a single tape, be sure you use the non-rewind device, if you use a rewind device (`rst0`) to back up multiple file systems, you'll end up overwriting the prior filesystem with the next one dump tries to write to tape. You can find a more elaborate description of various tape drive devices in the dump man page.

If you wanted to write a small script called "backup", it might look something like this:

```
echo "  Starting Full Backup..."
/sbin/dump -0au -f /dev/nrst0 /dev/rsd0a
/sbin/dump -0au -f /dev/nrst0 /dev/rsd0d
/sbin/dump -0au -f /dev/nrst0 /dev/rsd0e
/sbin/dump -0au -f /dev/nrst0 /dev/rsd0h
echo
echo -n "  Rewinding Drive, Please wait..."
mt -f /dev/rst0 rewind
echo "Done."
echo
```

If scheduled nightly backups are desired, `cron(8)` could be used to launch your backup script automatically.

It will also be helpful to document (on a scrap of paper) how large each file system needs to be. You can use "`df -h`" to determine how much space each partition is currently using. This will be handy when the drive fails and you need to recreate your partition table on the new drive.

Restoring your data will also help reduce fragmentation. To ensure you get all files, the best way of backing up is rebooting your system in single user mode. File systems do not need to be mounted to be backed up. Don't forget to mount root (/) r/w after rebooting in single user mode or your dump will fail when trying to write out dumpdates. Enter "`bsd -s`" at the boot> prompt for single user mode.

Viewing the contents of a dump tape:

After you've backed up your file systems for the first time, it would be a good idea to briefly test your tape and be sure the data on it is as you expect it should be.

You can use the following example to review a catalog of files on a dump tape:

```
# /sbin/restore -tvs 1 -f /dev/rst0
```

This will cause a list of files that exist on the 1st partition of the dump tape to be listed. Following along from the above examples, 1 would be your root (/) file system.

To see what resides on the 2nd tape partition and send the output to a file, you would use a command similar to:

```
# /sbin/restore -tvs 2 -f /dev/rst0 > /home/me/list.txt
```

If you have a mount table like the simple one, 2 would be `/usr`, if yours is a more advanced mount table 2 might be `/var` or another fs. The sequence number matches the order in which the file systems are written to tape.

Restoring from tape:

The example scenario listed below would be useful if your fixed drive has failed completely. In the event you want to restore a single file from tape, review the restore man page and pay attention to the interactive mode instructions.

If you have prepared properly, replacing a disk and restoring your data from tape can be a very quick process. The standard OpenBSD install/boot floppy already contains the required restore utility as well as the binaries required to partition and make your new drive bootable. In most cases, this floppy and your most recent dump tape is all you'll need to get back up and running.

After physically replacing the failed disk drive, the basic steps to restore your data are as follows:

- Boot from the OpenBSD install/boot floppy. At the menu selection, choose Shell. Write protect and insert your most recent back up tape into the drive.

- Using the fdisk(8) command, create a primary OpenBSD partition on this newly installed drive. Example:

  ```
  # fdisk -e sd0
  ```

  See "Using `fdisk(8)`" for more info.

- Using the disklabel command, recreate your OpenBSD partition table inside that primary OpenBSD partition you just created with fdisk. Example:

  ```
  # disklabel -E sd0
  ```

  (Don't forget swap, see "Using OpenBSD's `disklabel(8)`" for more info)

- Use the newfs command to build a clean file system on each partition you created in the above step. Example:

  ```
  # newfs /dev/rsd0a
  # newfs /dev/rsd0h
  ```

- Mount your newly prepared root (/) file system on **/mnt**. Example:

  ```
  # mount /dev/sd0a /mnt
  ```

- Change into that mounted root file system and start the restore process. Example:

  ```
  # cd /mnt
  # restore -rs 1 -f /dev/rst0
  ```

- You'll want this new disk to be bootable, use the following to write a new MBR to your drive. Example:

  ```
  # fdisk -i sd0
  ```

- In addition to writing a new MBR to the drive, you will need to install boot blocks to boot from it. The following is a brief example:

  ```
  # cp /usr/mdec/boot /mnt/boot
  # /usr/mdec/installboot -v /mnt/boot /usr/mdec/biosboot sd0
  ```

- Your new root file system on the fixed disk should be ready enough so you can boot it and continue restoring the rest of your file systems. Since your operating system is not complete yet, be sure you boot back up with single user mode. At the shell prompt, issue the following commands to unmount and halt the system:

```
# umount /mnt
# halt
```

- Remove the install/boot floppy from the drive and reboot your system. At the OpenBSD boot¿ prompt, issue the following command:

```
boot> bsd -s
```

  The `bsd -s` will cause the kernel to be started in single user mode which will only require a root (/) file system.

- Assuming you performed the above steps correctly and nothing has gone wrong you should end up at a prompt asking you for a shell path or press return. Press return to use sh. Next, you'll want to remount root in r/w mode as opposed to read only. Issue the following command:

```
# mount -u -w /
```

- Once you have re-mounted in r/w mode you can continue restoring your other file systems. Example:

```
(simple mount table)
# mount /dev/sd0h /usr; cd /usr; restore -rs 2 -f /dev/rst0

(more advanced mount table)
# mount /dev/sd0d /var; cd /var; restore -rs 2 -f /dev/rst0
# mount /dev/sd0e /home; cd /home; restore -rs 3 -f /dev/rst0
# mount /dev/sd0h /usr; cd /usr; restore -rs 4 -f /dev/rst0
```

  You could use "`restore rvsf`" instead of just rsf to view names of objects as they are extracted from the dump set.

- Finally after you finish restoring all your other file systems to disk, reboot into multiuser mode. If everything went as planned your system will be back to the state it was in as of your most recent back up tape and ready to use again.

14.11 Mounting disk images in OpenBSD

To mount a disk image (ISO images, disk images created with `dd`, etc.) in OpenBSD you must configure a `vnd(4)` device. For example, if you have an ISO image located at `/tmp/ISO.image`, you would take the following steps to mount the image.

```
# vnconfig vnd0 /tmp/ISO.image
# mount -t cd9660 /dev/vnd0c /mnt
```

Notice that since this is an ISO-9660 image, as used by CDs and DVDs, you must specify type of *cd9660* when mounting it. This is true, no matter what type, e.g. you must use type *ext2fs* when mounting Linux disk images.

To unmount the image use the following commands.

```
# umount /mnt
# vnconfig -u vnd0
```

For more information, refer to the `vnconfig(8)` man page.

14.12 Help! I'm getting errors with IDE DMA!

DMA IDE transfers, supported by `pciide(4)` are unreliable with many combinations of older hardware.

OpenBSD is aggressive and attempts to use the highest DMA Mode it can configure. This will cause corruption of data transfers in some configurations because of buggy motherboard chipsets, buggy drives, and/or noise on the cables. Luckily, Ultra-DMA modes protect data transfers with a CRC to detect corruption. When the Ultra-DMA CRC fails, OpenBSD will print an error message and try the operation again.

```
wd2a:  aborted command, interface CRC error reading fsbn 64 of 64-79
(wd2 bn 127; cn 0 tn 2 sn 1), retrying
```

After failing a couple times, OpenBSD will downgrade to a slower (hopefully more reliable) Ultra-DMA mode. If Ultra-DMA mode 0 is hit, then the drive downgrades to PIO mode.

UDMA errors are often caused by low quality or damaged cables. Cable problems should usually be the first suspect if you get many DMA errors or unexpectedly low DMA performance. It is also a bad idea to put the CD-ROM on the same channel with a hard disk.

If replacing cables does not resolve the problem and OpenBSD does not successfully downgrade, or the process causes your machine to lock hard, or causes excessive messages on the console and in the logs, you may wish to force the system to use a lower level of DMA or UDMA by default. This can be done by using `UKC` or `config(8)` to change the flags on the `wd(4)` device.

14.13 Why does df(1) tell me I have over 100% of my disk used?

People are sometimes surprised to find they have *negative* available disk space, or more than 100% of a filesystem in use, as shown by `df(1)`.

When a filesystem is created with `newfs(8)`, some of the available space is held in reserve from normal users. This provides a margin of error when you accidently fill the disk, and helps keep disk fragmentation to a minimum. Default for this is 5% of the disk capacity, so if the root user has been carelessly filling the disk, you may see up to 105

If the 5% value is not appropriate for you, you can change it with the `tunefs(8)` command.

14.14 Recovering partitions after deleting the disklabel

If you have a damaged partition table, there are various things you can attempt to do to recover it.

Firstly, panic. You usually do so anyways, so you might as well get it over with. Just don't do anything stupid. Panic away from your machine. Then relax, and see if the steps below won't help you out.

A copy of the disklabel for each disk is saved in `/var/backups` as part of the daily system maintenance. Assuming you still have the var partition, you can simply read the output, and put it back into disklabel.

In the event that you can no longer see that partition, there are two options. Fix enough of the disc so you can see it, or fix enough of the disc so that you can get your data off. Depending on what happened, one or other of those may be preferable (with dying discs you want the data first, with sloppy fingers you can just have the label).

The first tool you need is `scan_ffs(8)` (note the underscore, it isn't called "scanffs"). `scan_ffs(8)` will look through a disc, and try and find partitions and also tell you what information it finds about them. You can use this information to recreate the disklabel. If you just want `/var` back, you can recreate the partition for `/var`, and then recover the backed up label and add the rest from that.

`disklabel(8)` will update both the kernel's understanding of the disklabel, and then attempt to write the label to disk. Therefore, even if the area of the disk containing the disklabel is unreadable, you will be able to `mount(8)` it until the next reboot.

14.15 Can I access data on filesystems other than FFS?

Yes. Other supported filesystems include: ext2 (Linux), ISO9660 and UDF (CD-ROM, DVD media), FAT (MS-DOS and Windows), NFS, NTFS (Windows). Some of them have limited, for instance read-only, support.

We will give a general overview on how to use one of these filesystems under OpenBSD. To be able to use a filesystem, it must be mounted. For details and mount options, please consult the `mount(8)` manual page, and that of the mount command for the filesystem you will be mounting, e.g. `mount_msdos`, `mount_ext2fs`, ...

First, you must know on which device your filesystem is located. This can be simply your first hard disk, wd0 or sd0, but it may be less obvious. All recognized and configured devices on your system are mentioned in the output of the `dmesg(1)` command: a device name, followed by a one-line description of the device. For example, my first CD-ROM drive is recognized as follows:

```
cd0 at scsibus0 targ 0 lun 0: <COMPAQ, DVD-ROM LTD163, GQH3> SCSI0 5/cdrom removable
```

For a much shorter list of available disks, you can use `sysctl(8)`. The command

```
# sysctl hw.disknames
```

will show all disks currently known to your system, for example:

```
hw.disknames=cd0:,cd1:,wd0:,fd0:,cd2:
```

At this point, it is time to find out which partitions are on the device, and in which partition the desired filesystem resides. Therefore, we examine the device using `disklabel(8)`. The disklabel contains a list of partitions, with a maximum number of 16. Partition c always indicates the entire device. Partitions a-b and d-p are used by OpenBSD. Partitions i-p may be automatically allocated to file systems of other operating systems. In this case, I'll be viewing the disklabel of my hard disk, which contains a number of different filesystems.

NOTE: OpenBSD was installed after the other operating systems on this system, and during the install a disklabel containing partitions for the native as well as the foreign filesystems was installed on the disk. However, if you install foreign filesystems after the OpenBSD disklabel was already installed on the disk, you need to add or modify them manually afterwards. This will be explained in the "The partitions are not in my disklabel! What should I do?" subsection.

```
# disklabel wd0

# using MBR partition 2: type A6 off 20338290 (0x1365672) size 29318625 (0x1bf5de1)
# /dev/rwd0c:
```

```
type: ESDI
disk: ESDI/IDE disk
label: ST340016A
uid: d920a43a5a56ad5f
flags:
bytes/sector: 512
sectors/track: 63
tracks/cylinder: 16
sectors/cylinder: 1008
cylinders: 16383
total sectors: 78165360
boundstart: 20338290
boundend: 49656915
drivedata: 0

16 partitions:
#             size        offset  fstype [fsize bsize  cpg]
  a:        408366      20338290  4.2BSD   2048 16384   16 # /
  b:       1638000      20746656    swap
  c:      78165360             0  unused
  d:       4194288      22384656  4.2BSD   2048 16384   16 # /usr
  e:        409248      26578944  4.2BSD   2048 16384   16 # /tmp
  f:      10486224      26988192  4.2BSD   2048 16384   16 # /var
  g:      12182499      37474416  4.2BSD   2048 16384   16 # /home
  i:         64197            63 unknown
  j:      20274030         64260 unknown
  k:       1975932      49656978   MSDOS
  l:       3919797      51632973 unknown
  m:       2939832      55552833  ext2fs
  n:       5879727      58492728  ext2fs
  o:      13783707      64372518  ext2fs
```

As can be seen in the above output, the OpenBSD partitions are listed first. Next to them are a number of ext2 partitions and one MSDOS partition, as well as a few 'unknown' partitions. On i386 and amd64 systems, you can usually find out more about those using the `fdisk(8)` utility. For the curious reader: partition i is a maintenance partition created by the vendor, partition j is a NTFS partition and partition l is a Linux swap partition.

Once you have determined which partition it is you want to use, you can move to the final step: mounting the filesystem contained in it. Most filesystems are supported in the GENERIC kernel: just have a look at the kernel configuration file, located in the `/usr/src/sys/arch/<arch>/conf`directory. If you want to use one of the filesystems not supported in `GENERIC`, you will need to build a custom kernel.

When you have gathered the information needed as mentioned above, it is time to mount the filesystem. Let's assume a directory `/mnt/otherfs` exists,

which we will use as a mount point where we will mount the desired filesystem. In this example, we will mount the ext2 filesystem in partition m:

```
# mount -t ext2fs /dev/wd0m /mnt/otherfs
```

If you plan to use this filesystem regularly, you may save yourself some time by inserting a line for it in `/etc/fstab`, for example something like:

```
/dev/wd0m /mnt/otherfs ext2fs rw,noauto,nodev,nosuid 0 0
```

Notice the 0 values in the fifth and sixth field. This means we do not require the filesystem to be dumped, and checked using fsck. Generally, those are things you want to have handled by the native operating system associated with the filesystem.

14.15.1 The partitions are not in my disklabel! What should I do?

If you install foreign filesystems on your system (often the result of adding a new operating system) after you have already installed OpenBSD, a disklabel will already be present, and it will not be updated automatically to contain the new foreign filesystem partitions. If you wish to use them, you need to add or modify these partitions manually using `disklabel(8)`.

As an example, I have modified one of my existing ext2 partitions: using Linux's fdisk program, I've reduced the size of the 'o' partition (see disklabel output above) to 1G. We will be able to recognize it easily by its starting position (offset: 64372518) and size (13783707). Note that these values are sector numbers, and that using sector numbers (not megabytes or any other measure) is the most exact and safest way of reading this information.

Before the change, the partition looked like this using OpenBSD's `fdisk(8)` utility (leaving only relevant output):

```
# fdisk wd0
. . .
Offset: 64372455        Signature: 0xAA55
         Starting       Ending       LBA Info:
 #: id    C   H  S -    C   H  S [       start:      size   ]
------------------------------------------------------------------------
 0: 83 4007   1  1 - 4864 254 63 [    64372518:    13783707 ] Linux files*
. . .
```

As you can see, the starting position and size are exactly those reported by `disklabel(8)` earlier. (Dont' be confused by the value indicated by "Offset": it is referring to the starting position of the extended partition in which the ext2 partition is contained.)

After changing the partition's size from Linux, it looks like this:

```
# fdisk wd0
. . .
Offset: 64372455        Signature: 0xAA55
         Starting       Ending       LBA Info:
 #: id    C   H  S -    C   H  S [       start:      size   ]
------------------------------------------------------------------------
 0: 83 4007   1  1 - 4137 254 63 [    64372518:     2104452 ] Linux files*
. . .
```

Now this needs to be changed using `disklabel(8)`. For instance, you can issue `disklabel -e wd0`, which will invoke an editor specified by the `EDITOR` environment variable (default is `vi`). Within the editor, change the last line of the disklabel to match the new size:

```
o:       2104452      64372518  ext2fs
```

Save the disklabel to disk when finished. Now that the disklabel is up to date again, you should be able to mount your partitions as described above.

You can follow a very similar procedure to add new partitions.

14.16 Can I use a flash memory device with OpenBSD?

14.16.1 Flash memory as a portable storage device

Normally, the memory device should be recognized upon plugging it into a port of your machine. Shortly after inserting it, a number of messages are written to the console by the kernel. For instance, when I plug in my USB flash memory device, I see the following on my console:

```
umass0 at uhub1 port 1 configuration 1 interface 0
umass0: LEXR PLUG DRIVE LEXR PLUG DRIVE, rev 1.10/0.01, addr 2
umass0: using SCSI over Bulk-Only
scsibus2 at umass0: 2 targets
sd0 at scsibus2 targ 1 lun 0: <LEXAR, DIGITAL FILM, /W1.> SCSI2 0/direct removable
sd0: 123MB, 512 bytes/sec, 251904 sec total
```

These lines indicate that the `umass(4)` (USB mass storage) driver has been attached to the memory device, and that it is using the SCSI system. The last two lines are the most important ones: they are saying to which device node the memory device has been attached, and what the total amount of storage space is. If you somehow missed these lines, you can still see them afterwards with the `dmesg(1)` command. The reported CHS geometry is a rather fictitious one, as the flash memory is being treated like any regular SCSI disk.

We will discuss two scenarios below.

The device is new/empty and you want to use it with OpenBSD only

You will need to initialize a disklabel onto the device, and create at least one partition. Please read "Using OpenBSD's disklabel" and the `disklabel(8)` manual page for details about this.

In this example I created just one partition `a` in which I will place a FFS filesystem:

```
# newfs sd0a
Warning: inode blocks/cyl group (125) >= data blocks (62) in last
    cylinder group. This implies 1984 sector(s) cannot be allocated.
/dev/rsd0a:     249856 sectors in 122 cylinders of 64 tracks, 32 sectors
        122.0MB in 1 cyl groups (122 c/g, 122.00MB/g, 15488 i/g)
super-block backups (for fsck -b #) at:
 32,
```

Let's mount the filesystem we created in the `a` partition on `/mnt/flashmem`. Create the mount point first if it does not exist.

```
# mkdir /mnt/flashmem
# mount /dev/sd0a /mnt/flashmem
```

You received the memory device from someone with whom you want to exchange data

There is a considerable chance the other person is not using OpenBSD, so there may be a foreign filesystem on the memory device. Therefore, we will first need to find out which partitions are on the device, as described in "Can I access data on filesystems other than FFS?" section.

```
# disklabel sd0

# /dev/rsd0c:
type: SCSI
disk: SCSI disk
label: DIGITAL FILM
flags:
bytes/sector: 512
sectors/track: 32
tracks/cylinder: 64
sectors/cylinder: 2048
cylinders: 123
total sectors: 251904
rpm: 3600
interleave: 1
trackskew: 0
```

```
cylinderskew: 0
headswitch: 0           # microseconds
track-to-track seek: 0  # microseconds
drivedata: 0

16 partitions:
#             size        offset  fstype [fsize bsize  cpg]
  c:        251904             0  unused      0     0      # Cyl     0 -   122
  i:        250592            32   MSDOS                    # Cyl     0*-   122*
```

As can be seen in the disklabel output above, there is only one partition `i`, containing a FAT filesystem created on a Windows machine. As usual, the `c` partition indicates the entire device.

Let's now mount the filesystem in the `i` partition on `/mnt/flashmem`.

```
# mount -t msdos /dev/sd0i /mnt/flashmem
```

Now you can start using it just like any other disk.

WARNING: You should **always unmount** the filesystem **before unplugging** the memory device. If you don't, the filesystem may be left in an inconsistent state, which may result in data corruption.

Upon detaching the memory device from your machine, you will again see the kernel write messages about this to the console:

```
umass0: at uhub1 port 1 (addr 2) disconnected
sd0 detached
scsibus2 detached
umass0 detached
```

14.16.2 Flash memory as bootable storage

One can also use flash memory in various forms as bootable disk with OpenBSD. This can be done with both USB devices (assuming your computer can boot from a USB flash device, not all can), or with a non-USB (i.e., CF) device with an IDE or SATA adapter. (Non-USB devices attached with a USB adapter are treated as USB devices). In some cases, you may actually use a device in both ways (load the media in a USB adapter, but run it in an IDE adapter).

A flash device attached to a USB port will show up as a `sd(4)` SCSI-like device. When attached to an IDE adapter, it will show up as a `wd(4)` device.

In the case of flash media in an IDE adapter, it can be booted from any system that could boot from an IDE hard disk on the same adapter. In every sense, the system sees the flash media as an IDE disk. Simply configure the hardware appropriately, then install OpenBSD to the flash disk as normal.

In the case of booting from a USB device, your system must be able to boot from the USB device without being distracted by other devices on the system. Note that if your intention is to make a portable boot environment on a USB device, you really want to use DUIDs, rather than the traditional "`/dev/sd0X`"

notation. The USB device will show up as a SCSI disk, sometimes sd0. Without DUIDs, if you plug this device into a system which already has a few SCSI-like disks (i.e., devices attached to an `ahci(4)` interface) on it, it will probably end up with a different identifier, which will complicate carrying the flash device from system to system, as you would have to update `/etc/fstab`. Using DUIDs completely resolves this issue.

Some notes:

- **Speed**: In general, flash devices are much slower than hard disks, especially when it comes to writing. Using soft updates will help this considerably, as will using the "noatime" mount option.
- **"Write fatigue"**: Much has been written about the finite number of times an individual flash cell can be rewritten before failure. Practically speaking, however, there are many ways a flash device can fail, write fatigue is just one of them. Modern flash devices will verify writes, and in the event of failure, will automatically remap the failed sectors with one of the many spare sectors. Most users with most flash devices will not have to worry about "write fatigue". You would probably experience more down time due to failure of "clever" tricks done to avoid writing to the flash drive than you will by just using the drives as read-write media.
- **Reliability**: The fact that flash media has no moving parts has prompted many people to assume the flash media is inherently more reliable than hard disks. It is not wise to assume that switching to flash means you don't need to worry about data loss or drive failure. People have reported considerable variation in flash media quality, it is probably best to consider flash storage as a silent and low-power alternative to disk rather than a failure-free storage media.
- **Creating a bootable USB flash drive**: While a USB device can only be booted on a machine which can boot from USB drives, it can be created on any machine with supported USB hardware. You will, of course, be unable to test your work until you can get to a USB bootable system.
- **Going from IDE to USB interfaces**: Since flash media can be readable and writable through USB, IDE and other adapters, you can create bootable media with one type of adapter but maintain or use it with another type of adapter.
- **Mixing OpenBSD and other partitions on one device**: OpenBSD treats the flash disk as any other disk so one can use `fdisk(8)` to partition a flash device, as you would any hard disk. You can then have OpenBSD file systems on one partition, and use another partition for another file system, for example, FAT32. However, not all OSs treat USB devices as "equals". Windows, at least, will not attempt to use or create a partition that doesn't start at the beginning of the device, nor will the Windows partitioning tools allow you to partition the disk, though it will respect

existing partitions. So, if you wish to create a USB flash drive that is bootable with OpenBSD, but also functions as a FAT32-capable device on other OSs, you would want to do something like this:

1. Partition the media with OpenBSD's fdisk, creating a partition of the type you desire for Windows to use at the beginning of the disk, and an OpenBSD partition at the end of the disk.
2. Install OpenBSD as normal to the OpenBSD fdisk partition, don't forget to flag the OpenBSD partition as "Active" for booting.
3. Format the other partition. This can be (and perhaps should be) done on the "target" OS (Windows, in this case).

Note that if the other partition's type is chosen appropriately, it is possible to have OpenBSD access both partitions on the device. So, a Windows user could populate the FAT32 partition with MP3 files which could be played when they booted from the OpenBSD partition.

14.16.3 How do I create a bootable "Live" USB device?

It is very easy to create a bootable USB flash (or other!) drive that can be used as a "live" OpenBSD system without installing OpenBSD on the local hard disk of a machine. Obviously, the target machine must be bootable from a USB device, but the initial load can actually be done from any machine with a USB interface.

Some reasons you may want to do this:

- A portable, secure "machine" you can carry with you in your pocket.
- Fix problems in OpenBSD and other installs with tools that aren't available with bsd.rd
- Evaluate hardware for OpenBSD compatibility at the point of purchase.
- Install machines that lack any other form of boot media.
- Collect dmesg outputs from your friend's computers! (As OpenBSD's dmesg is useful for identifying hardware, this can be a good way to prep for the install of OTHER OSs on a machine – make sure you have all the needed drivers before starting).

Creating such a "live OpenBSD drive" is simple.

- Mount your USB drive to the machine you are installing from.
- Boot your favorite OpenBSD install media.
- Install as normal, being careful to select your flash drive as the install target.

- Boot from your newly created USB device.

That's it!

There are some things you may want to do after the install to improve your results:

- Install all packages and utilities you are likely to wish to have available.
- Different target machines will likely have different NICs. You could create a bunch of `hostname.if(5)` files in `/etc`, each containing just `dhcp`, for all the NICs you are likely to encounter (`fxp0`, `re0`, `rl0`, `bge0`, `bnx0`, `em0`, etc.) on USB-bootable machines, plus maybe sample wireless config files as well. OpenBSD will ignore all `hostname.if(5)` files for devices not present at boot time.
- You may wish to have a copy of the install files and maybe any desired package `.tgz` files so you can use the drive as an install media (boot `bsd.rd` instead of letting it boot normally).
- On many machines, X will "Just Work" with no config file, but you may wish to collect X config files for those systems that require them.
- Soft updates will be something you will want to use.
- For maximum flexibility, you probably want to use i386 rather than amd64. However, if you wish to use it as an install media, you could have an amd64 bsd.rd and install files in addition to i386.
- Use DUIDs to identify your partitions.
- You may find it desirable to have a FAT partition on the USB drive, create it as detailed above.
- You may wish to use `softraid(4)` to encrypt a data partition.

14.17 Optimizing disk performance

Disk performance is a significant factor in the overall speed of your computer. It becomes increasingly important when your computer is hosting a multi-user environment (users of all kinds, from those who log-in interactively to those who see you as a file-server or a web-server). Data storage constantly needs attention, especially when your partitions run out of space or when your disks fail. OpenBSD has a few options to increase the speed of your disk operations.

- Soft Updates
- Size of the `namei()` cache

14.17.1 Soft updates

An important tool that can be used to speed up your system is softupdates. One of the slowest operations in the traditional BSD file system is updating metainfo (which happens, among other times, when you create or delete files and directories). Softupdates attempts to update metainfo in RAM instead of writing to the hard disk each and every single metainfo update. Another effect of this is that the metainfo on disk should always be complete, although not always up to date. You can read more about softupdates in the Softupdates entry.

14.17.2 Size of the `namei()` cache

The name-to-inode translation (a.k.a., `namei()`) cache controls the speed of pathname to `inode(5)` translation. A reasonable way to derive a value for the cache, should a large number of `namei()` cache misses be noticed with a tool such as `systat(1)`, is to examine the system's current computed value with `sysctl(8)`, (which calls this parameter "`kern.maxvnodes`") and to increase this value until either the `namei()` cache hit rate improves or it is determined that the system does not benefit substantially from an increase in the size of the `namei()` cache. After the value has been determined, you can set it at system startup time with `sysctl.conf(5)`.

14.18 Why aren't we using async mounts?

Question: "I simply do "`mount -u -o async /`" which makes one package I use (which insists on touching a few hundred things from time to time) usable. Why is async mounting frowned upon and not on by default (as it is in some other unixen)? Isn't it a much simpler, and therefore, a safer way of improving performance in some applications?"

Answer: "Async mounts are indeed faster than sync mounts, but they are also less safe. What happens in case of a power failure? Or a hardware problem? The quest for speed should not sacrifice the reliability and the stability of the system. Check the man page for `mount(8)`."

> async
> All I/O to the file system should be done asynchronously. This is a dangerous flag to set since it does not guaran- tee to keep a consistent file system structure on the disk. You should not use this flag unless you are pre- pared to recreate the file system should your system crash. The most common use of this flag is to speed up restore(8) where it can give a factor of two speed in- crease.

On the other hand, when you are dealing with temp data that you can recreate from scratch after a crash, you can gain speed by using a separate partition for that data only, mounted async. Again, do this *only if* you don't

mind the loss of all the data in the partition when something goes wrong. For this reason, `mfs(8)` partitions are mounted asynchronously, as they will get wiped and recreated on a reboot anyway.

Chapter 15

The OpenBSD packages and ports system

15.1 Introduction

There are a lot of third party applications available which one might want to use on an OpenBSD system. To make this software easier to install and manage, plus to help it comply with OpenBSD's policy and goals, the third party software is *ported* to OpenBSD. This porting effort can involve many different things. Examples are: making the software use the standard OpenBSD directory layout (e.g. configuration files go into `/etc`), conforming to OpenBSD's shared library specifications, making the software more secure whenever possible, etc.

The end result of the porting effort are ready-to-install binary packages. The aim of the package system is to keep track of which software gets installed, so that it may at any time be updated or removed very easily. This way, no unnecessary files are left behind, and users can keep their systems clean. The package system also helps ensure nothing is deleted by accident, causing software to stop functioning properly. Another advantage is that **users rarely need to compile software from source**, as packages have already been compiled and are available and ready to be used on an OpenBSD system. In minutes, a large number of packages can be fetched and installed, with everything in the right place.

The packages and ports collection does NOT go through the same thorough security audit that is performed on the OpenBSD base system. Although we strive to keep the quality of the packages collection high, we just do not have enough human resources to ensure the same level of robustness and security. Of course security updates for various applications are committed to the ports tree as soon as possible, and corresponding package security updates are made available as snapshots for *-current*.

15.2 Package Management

15.2.1 How does it work?

Packages are the pre-compiled binaries of some of the most used third party software. Packages can be managed easily with the help of several utilities, also referred to as the pkg* tools:

- `pkg_add(1)` - a utility for installing and upgrading software packages.
- `pkg_delete(1)` - a utility for deleting previously installed software packages.
- `pkg_info(1)` - a utility for displaying information about software packages.
- `pkg_create(1)` - a utility for creating software packages.

In order to run properly, an application X may require that other applications Y and Z be installed. Application X is said to be dependent on these other applications, which is why Y and Z are called *dependencies* of X. In turn, Y

may require other applications P and Q, and Z may require application R to function properly. This way, a whole *dependency* tree is formed.

Packages look like simple `.tgz` bundles. Basically they are just that, but there is one crucial difference: they contain some extra *packing information*. This information is used by `pkg_add(1)` for several purposes:

- Different checks: has the package already been installed or does it conflict with other installed packages or file names?
- Dependencies which are not yet present on the system, are automatically fetched and installed, before proceeding with the installation of the package.
- Information about the package(s) is recorded in a central repository, by default located in `/var/db/pkg/`. This will, among other things, prevent the dependencies of a package from being deleted before the package itself has been deleted. This helps ensure that an application cannot be accidentally broken by a careless user.

15.2.2 Making things easy: PKG_PATH

You can make things really easy by using the `PKG_PATH` environment variable. Just point it to your favorite location, and `pkg_add(1)` will automatically look there for any package you specify, `and` also fetch and install the necessary dependencies of this package automatically.

A list of possible locations to fetch packages from is given in the following section.

Example 1: fetching from your CDROM, assuming you mounted it on `/mnt/cdrom`

```
$ export PKG_PATH=/mnt/cdrom/5.0/packages/‘machine -a‘/
```

Example 2: fetching from a nearby FTP mirror

```
$ export PKG_PATH=ftp://your.ftp.mirror/pub/OpenBSD/5.0/packages/‘machine -a‘/
```

It's usually a good idea to add a line similar to the above examples to your `/.profile`. As with the classic PATH variable, you can specify multiple locations, separated by colons. **Prior to OpenBSD 4.4, every path in the `PKG_PATH` variable MUST end in a slash (/).** That way, `pkg_add(1)` can split the path correctly even if it holds URL schemes containing colons. If the first entry in `PKG_PATH` fails, the next one will be tried, and so on, until the package is found. If all entries fail, an error is produced.

Notice the use of `machine(1)` in the above command lines. This automatically substitutes your installed OpenBSD "application architecture", which is usually, but not always, your platform name. Of course, if you are using snapshots, you will replace "5.0" with "snapshots".

15.2.3 Finding packages

A large collection of pre-compiled packages is available for the most common architectures. Just look for your package in one of these places:

- On one of the three CD-ROMs, depending on your architecture. The CD-ROMs carry only the most commonly used, freely distributable packages for the most commonly used platforms.
- On the FTP mirror servers. Packages are located in the `/pub/OpenBSD/5.0/packages` directory. From there, packages are broken down depending on architecture.

If you have the ports tree on your system, you can quickly find the package you are looking for as explained in Searching the ports tree.

You will notice that certain packages are available in a few different varieties, formally called **flavors**. Others are pieces of the same application which may be installed separately. They are called **subpackages**. This will be detailed further in "Using flavors and subpackages" but flavor basically means they are configured with different sets of options. Currently, many packages have flavors, for example: database support, support for systems without X, or network additions like SSL and IPv6. Every flavor of a package will have a different suffix in its package name. For detailed information about package names, please refer to `packages-specs(7)`.

Note: Not all possible packages are necessarily available on the FTP servers! Some applications simply don't work on all architectures. Some applications can not be distributed via FTP (or CDROM) for licensing reasons. There may also be many possible combinations of flavors of a port, and the OpenBSD project just does not have the resources to build them all. If you need a combination which is not available, you will have to build the port from source. For more information on how to do that, read Using flavors and subpackages in the Ports section of this document.

15.2.4 Installing new packages

To install packages, the utility `pkg_add(1)` is used. If you have made things easy for yourself by setting the `PKG_PATH` environment variable, you can just call `pkg_add(1)` with the package name, as in the following basic example.

```
$ sudo pkg_add -v screen-4.0.3p1
parsing screen-4.0.3p1
installed /etc/screenrc from /usr/local/share/examples/screen/screenrc | 71%
screen-4.0.3p1: complete
```

In this example the `-v` flag was used to give a more verbose output. This option is not needed but it is helpful for debugging and was used here to give a little more insight into what `pkg_add(1)` is actually doing. Notice the message mentioning `/etc/screenrc`. Specifying multiple `-v` flags will produce even more verbose output.

Using pkg_add(1) in interactive mode

Since OpenBSD 3.9, pkg_add(1) has an interactive mode, which is enabled by invoking it with the -i flag, and which causes it to ask you questions when it cannot make decisions by itself. For example, if you don't know the version number of a package beforehand, you can try something like:

```
$ sudo pkg_add -i screen
Ambiguous: screen could be screen-4.0.3p1 screen-4.0.3p1-shm screen-4.0.3p1-static
Choose one package
         0: <None>
         1: screen-4.0.3p1
         2: screen-4.0.3p1-shm
         3: screen-4.0.3p1-static
Your choice: 1
screen-4.0.3p1: complete
```

For some packages, some important additional information will be given about the configuration or use of the application on an OpenBSD system. Since it is important, it will be displayed whether or not you use the -v flag. Consider the following example:

```
$ sudo pkg_add ghostscript-fonts-8.11
ghostscript-fonts-8.11: complete
You may wish to update your font path for /usr/local/share/ghostscript/fonts
--- ghostscript-fonts-8.11 -------------------
To install these fonts for X11, just make sure that the fontpath
lists the 75dpi or 100dpi bitmap fonts before the ghostscript fonts,
and make sure you have the string ":unscaled" appended to the bitmap
font's fontpath. This way, the bitmap fonts will be used if they
match, and the Type 1 versions will be used if the font needs to be
scaled. Below is the relevant section from a typical xorg.conf file.

   FontPath   "/usr/X11R6/lib/X11/fonts/misc/"
   FontPath   "/usr/X11R6/lib/X11/fonts/75dpi/:unscaled"
   FontPath   "/usr/X11R6/lib/X11/fonts/100dpi/:unscaled"
   FontPath   "/usr/local/lib/X11/fonts/ghostscript/"
   FontPath   "/usr/X11R6/lib/X11/fonts/Type1/"
```

Let us now continue with an example of a package which has dependencies:

```
$ sudo pkg_add -v tin-1.8.2p0
parsing tin-1.8.2p0
Dependencies for tin-1.6.2 resolve to: gettext-0.14.6, libutf8-0.8, pcre-6.4p1, libiconv-1.9.2p3
tin-1.8.2p0:parsing libiconv-1.9.2p3
tin-1.8.2p0:libiconv-1.9.2p3: complete
tin-1.8.2p0:parsing gettext-0.14.6
```

```
Dependencies for gettext-0.14.6 resolve to: expat-2.0.0, libiconv-1.9.2p3 (todo: expat
tin-1.8.2p0:parsing expat-2.0.0
tin-1.8.2p0:expat-2.0.0: complete
tin-1.8.2p0:gettext-0.14.6: complete
tin-1.8.2p0:parsing pcre-6.4p1
tin-1.8.2p0:pcre-6.4p1: complete
tin-1.8.2p0:parsing libutf8-0.8
tin-1.8.2p0:libutf8-0.8: complete
tin-1.8.2p0: complete
```

Again we added the `-v` flag to see more of what is happening. Upon investigating the packing information, dependencies are found and they are installed first. Somewhere in the middle you can see the gettext package being installed, which depends on libiconv. Before installing gettext, its packing information is examined and it is verified whether libiconv has already been installed.

It is possible to specify multiple package names on one line, which then all get installed at once, along with possible dependencies.

If for some reason you decide not to use `PKG_PATH`, it is also possible to specify the absolute location of a package on the command line. This absolute location may be a local path, or a URL referring to FTP, HTTP, or SCP locations. Let's consider installation via FTP in the next example:

```
$ sudo pkg_add ftp://ftp.openbsd.org/pub/OpenBSD/5.0/packages/‘machine -a‘/screen-4.0.3
screen-4.0.3p1: complete
```

In this example the `-v` flag wasn't used, so only needed messages are shown. Notice that the complete filename was entered by adding a `.tgz` suffix. You can also skip this suffix as in the previous examples since it is auto-completed by `pkg_add(1)`.

Note: Not all architectures have the same packages available. Some ports do not work on certain architectures, and performance limits the number of packages that can be built on others.

For safety, if you are installing a package which you had installed earlier (or an older version of it) and removed, `pkg_add(1)` will not overwrite configuration files which have been modified. Instead, it will inform you about this as follows (only when using the `-v` flag, however!):

```
$ sudo pkg_add -v screen-4.0.3p1
parsing screen-4.0.3p1
The existing file /etc/screenrc has NOT been changed**                  | 71%
It does NOT match the sample file /usr/local/share/examples/screen/screenrc
You may wish to update it manually
screen-4.0.3p1: complete
```

Sometimes you may encounter an error like the one in the following example:

```
$ sudo pkg_add xv-3.10ap4
```

```
xv-3.10ap4:jpeg-6bp3: complete
xv-3.10ap4:png-1.2.14p0: complete
xv-3.10ap4:tiff-3.8.2p0: complete
Can't install xv-3.10ap4: lib not found X11.9.0
Even by looking in the dependency tree:
        tiff-3.8.2p0, jpeg-6bp3, png-1.2.14p0
Maybe it's in a dependent package, but not tagged with @lib ?
(check with pkg_info -K -L)
If you are still running 3.6 packages, update them.
```

There is `pkg_add(1)` nicely installing dependencies, when all of a sudden it aborts the installation of xv. This is another safety precaution which is available since OpenBSD 3.7. The packing information bundled in the package includes information about shared libraries that the package expects to be installed, system libraries as well as third party libraries. If one of the required libraries cannot be found, the package is not installed because it would not function anyway.

To solve this type of conflict, you must find out what to install in order to get the required libraries on your system. There are several things to check:

- You may have older packages installed: an older version of the required library is present. In this case, upgrade these packages.
- Your system may be incomplete: you did not install one of the file sets, which contains the required library. Just add the required file set.
- Your system may be out of date: you have an older version of the required library. Boot the installer (as detailed in FAQ 4), and choose to (U)pgrade your complete system.

15.2.5 Listing installed packages

You can see a list of installed packages by using the `pkg_info(1)` utility.

```
$ pkg_info
aterm-0.4.2p1       color vt102 terminal emulator with transparency support
bzip2-1.0.4         block-sorting file compressor, unencumbered
expat-2.0.0         XML 1.0 parser written in C
fluxbox-0.9.15.1p0  window manager based on the original Blackbox code
gettext-0.14.6      GNU gettext
imlib2-1.3.0        image manipulation library
jpeg-6bp3           IJG's JPEG compression utilities
libiconv-1.9.2p3    character set conversion library
libltdl-1.5.22p1    GNU libtool system independent dlopen wrapper
libungif-4.1.4p0    tools and library routines for working with GIF images
libutf8-0.8         provides UTF-8 locale support
mutt-1.4.2.2i       tty-based e-mail client
```

```
pcre-6.4p1          perl-compatible regular expression library
png-1.2.14p0        library for manipulating PNG images
screen-4.0.3p1      multi-screen window manager
tcsh-6.14.00p1      extended C-shell with many useful features
tiff-3.8.2p0        tools and library routines for working with TIFF images
tin-1.8.2p0         threaded NNTP and spool based UseNet newsreader
```

When given an installed package name (or a location of a package which is to be installed), `pkg_info(1)` will show more detailed information about that specific package.

15.2.6 Updating installed packages

Let's say you had an older version of unzip installed before upgrading this box from OpenBSD 4.9 to 5.0. Now you can easily upgrade to the newer 5.0 package as follows:

```
$ sudo pkg_add -u unzip
unzip-5.52p0 (extracting): complete
unzip-5.52 (deleting): complete
unzip-5.52p0 (installing): complete
Clean shared items: complete
```

When a package has dependencies, they are also examined for updates. Invoking `pkg_add(1)` with the `-u` flag and no package name will try to update all installed packages.

Note: The `-u` switch relies on the `PKG_PATH` environment variable. If it is not set, `pkg_add(1)` will not be able to find updates.

Having several entries in `PKG_PATH` does not mean all entries will be tried for update operations. Instead, `pkg_add(1)` will stop at the first path with matching candidates.

If you had a configuration file belonging to the old version, which you modified, it will be left untouched by default. You can, however, replace it with the default configuration file of the new version, by calling `pkg_add(1)` with the `-c` flag.

15.2.7 Removing installed packages

To delete a package, simply take the proper name of the package as shown by `pkg_info(1)` (see Listing installed packages above) and use `pkg_delete(1)` to remove the package. In the example below, the screen package is being removed. Notice that on some occasions there are instructions of extra items that need to be removed that `pkg_delete(1)` did not remove for you. As with the `pkg_add(1)` utility, you can use the `-v` flag to get more verbose output.

```
$ sudo pkg_delete screen
screen-4.0.3p1: complete
Clean shared items: complete
```

Note: Often, it is not necessary to specify the version numbers and flavors with the package name, since `pkg_delete(1)` will usually be able to find the full name by itself. You need to specify the complete package name (in the example, that is screen-4.0.3p1) only if ambiguity is possible due to multiple installed packages with the specified name. In that case `pkg_delete(1)` cannot know which package to delete.

For safety, `pkg_delete(1)` will not remove configuration files if they have been modified. Instead it will inform you about this as follows:

```
$ sudo pkg_delete screen
screen-4.0.3p1: complete
Clean shared items: complete
--- screen-4.0.3p1 -------------------
You should also remove /etc/screenrc (which was modified)
```

If you prefer to have those configuration files removed automatically, you can do so by using the `-c` flag.

15.2.8 Incomplete package installation or removal

In some odd cases, you may find that a package was not added or deleted completely, because of conflicts with other files. The incomplete installation is usually marked with "partial-" prepended to the package name. This can, for instance, happen when you coincidentally press CTRL+C during installation:

```
$ sudo pkg_add screen-4.0.3p1
screen-4.0.3p1: complete                                                    7%
Adjusting md5 for /usr/local/info/screen.info-3 from 49fb3fe1cc3a3b0057518459811b6dac to 3b9c7811
/usr/sbin/pkg_add: Installation of screen-4.0.3p1 failed , partial installation recorded as parti
```

It is always a good idea to remove partial packages from your system, and to fix potential problems that lead to this failure. It is often an indication that you do not have a clean system with everything installed from packages, but possibly packages mixed up with other software installed straight from source.

15.3 Working with ports

As mentioned in the introduction, packages are compiled from the ports tree. In this section we will explain how the ports tree works, when you should use it and how you can use it.

IMPORTANT NOTE: The ports tree is meant for advanced users. **Everyone is encouraged to use the pre-compiled binary packages**. Do NOT ask beginner questions on the mailing lists like "How can I get the ports tree working?". If you have questions about the ports tree, it is assumed that you have read the manual pages and this FAQ, and that you are able to work with it.

15.3.1 How does it work?

The ports tree, a concept originally borrowed from FreeBSD, is a set of Makefiles, one for each third party application, for controlling

- where and how to fetch the source of the software,
- which other software it depends upon,
- how to alter the sources (if necessary),
- how to configure and build it,
- how to test it (optional),
- how to install it.

Apart from the Makefile, each port also contains at least the following:

- a PLIST or packing list, which contains instructions for package creation once the application has been built,
- a DESCR or description of the application,
- a distfile, containing distribution file checksums and size

All this information is kept in a directory hierarchy under `/usr/ports`. This hierarchy contains three special subdirectories:

- `distfiles/` - where the ports system stores software distribution sets after downloading.
- `infrastructure/` - the main directory of the ports infrastructure, containing all necessary scripts and makefiles.
- `packages/` - contains all binary packages built by the ports system.

The other subdirectories all form different application categories, which contain the subdirectories of the actual ports. Complex ports may be organized to an even deeper level, for example if they have a core part and a set of extensions, or a stable and a snapshot version of the application. Every port directory must contain a `pkg/` subdirectory containing packing list(s) and description file(s). There may also be `patches/` and `files/` subdirectories, for source patches and additional files, respectively.

When a user issues `make(1)` in the subdirectory of a specific port, the system will recursively walk its dependency tree, check whether the required dependencies are installed, build and install any missing dependencies, and then continue the build of the desired port. All of the building happens inside the *working directory* that the port creates. Normally it is under `${WRKOBJDIR}`, defaulting to `/usr/ports/pobj`, but you may override this (see Configuration of the ports

system). If `WRKOBJDIR` has been explicitly unset, a subdirectory of the port's main directory (package name prefixed by "w-") will be used instead.

Note: Ports are never directly installed on your system! They use a *fake installation directory*. Everything that gets installed there, is bundled together into a package (which is stored in the `packages/` subdirectory of the ports tree as mentioned earlier). Installing a port really means: creating a package, and then installing that package!

More information about the ports system may be found in these manual pages:

- `ports(7)` - describes the different stages (make targets) of port installation, the use of flavors and subpackages and some other options.
- `bsd.port.mk(5)` - in depth information about all the make targets, variables, the fake (installation directory) framework, etc.

15.3.2 Fetching the ports tree

Before continuing, you must read the section about NOT mixing up your OpenBSD system and ports tree. Once you have decided which flavor of the ports tree you want, you can get the ports tree from different sources. The table below gives an overview of where you can find the different flavors, and in which form. An 'x' marks availability and '-' means it is not available through that specific source.

Source	Form	Flavour			
		-release	-stable	snapshots	-current
CD-ROM	.tar.gz	x	-	-	-
FTP mirrors	.tar.gz	x	-	x	-
AnonCVS	cvs checkout	x	x	-	x

On the CD-ROM and FTP mirrors, look for a file named `ports.tar.gz`. You want to untar this file in the `/usr` directory, which will create `/usr/ports`, and all the directories under it. For example:

```
$ cd /tmp
$ ftp ftp://ftp.openbsd.org/pub/OpenBSD/5.0/ports.tar.gz
$ cd /usr
$ sudo tar xzf /tmp/ports.tar.gz
```

The snapshots available on the FTP mirrors are generated daily from the -current ports tree. You will find the snapshots in the `pub/OpenBSD/snapshots/` directory. If you are using a snapshot of the ports tree, you should have installed a matching snapshot of OpenBSD. Make sure you keep your ports tree and your OpenBSD system in sync!

For more information about obtaining the ports tree via AnonCVS, please read the AnonCVS page which contains a list of available servers and a number of examples.

15.3.3 Configuration of the ports system

NOTE: This section introduces some additional global settings for building applications from ports. You can skip this section, but then you will be required to perform many of the `make(1)` statements in the examples as root.

Because the OpenBSD project does not have the resources to fully review the source code of all software in the ports tree, you can configure the ports system to take a few safety precautions. The ports infrastructure is able to perform all building as a regular user, and perform only those steps that require superuser privileges as root. Examples are the `fake` and `install` make targets. However, because root privileges are always required at some point, the ports system will not save you when you decide to build a malicious application.

- You can set up `sudo(8)` and have the ports system use it for tasks requiring superuser permissions. Just add a line to `/etc/mk.conf` containing

  ```
  SUDO=/usr/bin/sudo
  ```

- You can modify the ownerships of the ports tree so that you can write there as a regular user. In this case, the regular user has been added to the wsrc group, and the underlying directories are made group writable.

  ```
  # chgrp -R wsrc /usr/ports
  # find /usr/ports -type d -exec chmod g+w {} \;
  ```

- You can have the ports system use `systrace(1)` by adding the following to `/etc/mk.conf`

  ```
  USE_SYSTRACE=Yes
  ```

  This enforces the build procedure to stay inside allowed directories, and prohibits writing in illegal places, thereby considerably reducing the risk of a damaged system. Note that the use of `systrace(1)` adds about 20% overhead in build time.

It is possible to use a read-only ports tree by separating directories that are written to during port building:

- The working directory of ports. This is controlled by the `WRKOBJDIR` variable, which specifies the directory which will contain the working directories.
- The directory containing distribution files. This is controlled by the `DISTDIR` variable.
- The directory containing newly built binary packages. This is controlled by the `PACKAGE_REPOSITORY` variable.

For example, you could add the following lines to /etc/mk.conf

```
WRKOBJDIR=/usr/obj/ports
DISTDIR=/usr/distfiles
PACKAGE_REPOSITORY=/usr/packages
```

If desired, you can also change the ownership of these directories to your local username and group, so that the ports system can create underlying working directories as a regular user.

15.3.4 Searching the ports tree

Once you have the ports tree in place on your system, it becomes very easy to search for software. Just use make search key="searchkey", as shown in the following example.

```
$ cd /usr/ports
$ make search key=rsnapshot
Port:   rsnapshot-1.3.1p0
Path:   net/rsnapshot
Info:   remote filesystem snapshot utility
Maint:  Antoine Jacoutot <ajacoutot@openbsd.org>
Index:  net sysutils
L-deps:
B-deps: :net/rsync
R-deps: :devel/p5-Lchown :net/rsync
Archs:  any
```

The search result gives a nice overview of each application that is found: the port name, the path to the port, a one-line description, the port's maintainer, keywords related to the port, library/build/runtime dependencies, and architectures on which the port is known to work.

This mechanism, however, is a very basic one, which just runs awk(1) on the ports index file. Since OpenBSD 4.0, a new port called "sqlports" has been created, allowing very fine-grained searching using SQL. It is a SQLite database, but basically just about any database format can be created using the ports infrastructure. The sqlports port includes the script used to generate the database, which could be used as a basis to generate databases in different formats.

Just pkg_add(1) the sqlports package, and in this case, the sqlite3 package to get started. A sample session could look like:

```
$ sqlite3 /usr/local/share/sqlports
SQLite version 3.3.12
Enter ".help" for instructions
sqlite> SELECT FULLPKGNAME,COMMENT FROM Ports WHERE COMMENT LIKE '%statistics%';
Guppi-0.40.3p1|GNOME-based plot program with statistics capabilities
```

```
mailgraph-1.12|a RRDtool frontend for Postfix statistics
R-2.4.1|clone of S, a powerful math/statistics/graphics language
py-probstat-0.912p0|probability and statistics utilities for Python
darkstat-3.0.540p1|network statistics gatherer with graphs
pfstat-2.2p0|packet filter statistics visualization
tcpstat-1.4|report network interface statistics
wmwave-0.4p2|Window Maker dockapp to display wavelan statistics
diffstat-1.43p0|accumulates and displays statistics from a diff file
sqlite>
```

The above is still a very basic search. With SQL, just about anything can be searched for, including dependencies, configure flags, shared libraries, etc.

15.3.5 Straightforward installation: a simple example

For clarity's sake, let's consider a simple example: rsnapshot. This application has one dependency: rsync.

```
$ cd /usr/ports/net/rsnapshot
$ make install
===>  Checking files for rsnapshot-1.2.9
>> rsnapshot-1.2.9.tar.gz doesn't seem to exist on this system.
>> Fetch http://www.rsnapshot.org/downloads/rsnapshot-1.2.9.tar.gz.
100% |**************************************************|   173 KB    00:02
>> Size matches for /usr/ports/distfiles/rsnapshot-1.2.9.tar.gz
>> Checksum OK for rsnapshot-1.2.9.tar.gz. (sha1)
===>  rsnapshot-1.2.9 depends on: rsync-2.6.9 - not found
===>  Verifying install for rsync-2.6.9 in net/rsync
===>  Checking files for rsync-2.6.9
>> rsync-2.6.9.tar.gz doesn't seem to exist on this system.
>> Fetch ftp://ftp.samba.org/pub/rsync/old-versions/rsync-2.6.9.tar.gz.
100% |**************************************************|   792 KB    00:31
>> Size matches for /usr/ports/distfiles/rsync-2.6.9.tar.gz
>> Checksum OK for rsync-2.6.9.tar.gz. (sha1)
===>  Verifying specs:  c
===>  found c.40.3
===>  Extracting for rsync-2.6.9
===>  Patching for rsync-2.6.9
===>  Configuring for rsync-2.6.9
  [...snip...]
===>  Building for rsync-2.6.9
  [...snip...]
===>  Faking installation for rsync-2.6.9
  [...snip...]
===>  Building package for rsync-2.6.9
Link to /usr/ports/packages/i386/ftp/rsync-2.6.9.tgz
```

```
Link to /usr/ports/packages/i386/cdrom/rsync-2.6.9.tgz
===>  Installing rsync-2.6.9 from /usr/ports/packages/i386/all/rsync-2.6.9.tgz
rsync-2.6.9: complete
===> Returning to build of rsnapshot-1.2.9
===>  rsnapshot-1.2.9 depends on: rsync-2.6.9 - found
===>  Extracting for rsnapshot-1.2.9
===>  Patching for rsnapshot-1.2.9
===>  Configuring for rsnapshot-1.2.9
  [...snip...]
===>  Building for rsnapshot-1.2.9
  [...snip...]
===>  Faking installation for rsnapshot-1.2.9
  [...snip...]
===>  Building package for rsnapshot-1.2.9
Link to /usr/ports/packages/i386/ftp/rsnapshot-1.2.9.tgz
Link to /usr/ports/packages/i386/cdrom/rsnapshot-1.2.9.tgz
===>  rsnapshot-1.2.9 depends on: rsync-2.6.9 - found
===>  Installing rsnapshot-1.2.9 from /usr/ports/packages/i386/all/rsnapshot-1.2.9.tgz
rsnapshot-1.2.9: complete
```

As you can see, the ports system is doing many things automatically. It will fetch, extract, and patch the source code, configure and build (compile) the source, install the files into a fake directory, create a package (corresponding to the packing list) and install this package onto your system (usually under **/usr/local/**). And it does this recursively **for all dependencies** of the port. Just notice the "`===> Verifying install for ...`" and "`===> Returning to build of ...`" lines in the above output, indicating the walk through the dependency tree.

If a previous version of the application you want to install, was already installed on your system, you can use `make update` instead of `make install`. This will call `pkg_add(1)` with the `-r` flag.

Note: Large applications will require a lot of system resources to build. Good examples are compilers like GCC 4.0 or the Java 2 Software Development Kit. If you get "out of memory" type of errors when building such a port, this usually has one of two causes:

- Your resource limits are too restrictive. Adjust them with ksh's ulimit or csh's limit command. If that doesn't help, just become root before starting the build, or use `sudo(8)` with the `-c` flag to run the build with the resources limited by the specified login class (refer to `login.conf(5)` for details about login classes).

- You simply don't have enough RAM in your machine.

15.3.6 Cleaning up after a build

You probably want to clean the port's default working directory after you have built the package and installed it.

```
$ make clean
===>  Cleaning for rsnapshot-1.2.9
```

In addition, you can also clean the working directories of all dependencies of the port with this make target:

```
$ make clean=depends
===>  Cleaning for rsync-2.6.9
===>  Cleaning for rsnapshot-1.2.9
```

If you wish to remove the source distribution set(s) of the port, you would use

```
$ make clean=dist
===>  Cleaning for rsnapshot-1.2.9
===>  Dist cleaning for rsnapshot-1.2.9
```

In case you have been compiling multiple flavors of the same port, you can clear the working directories of all these flavors at once using

```
$ make clean=flavors
```

You can also clean things up as they get built, by setting a special variable. Work directories will automatically be cleaned after packages have been created:

```
$ make package BULK=Yes
```

15.3.7 Uninstalling a port's package

It is very easy to uninstall a port:

```
$ make uninstall
===> Deinstalling for rsnapshot-1.2.9
rsnapshot-1.2.9: complete
Clean shared items: complete
```

This will call `pkg_delete(1)` to have the corresponding package removed from your system. If desired, you can also uninstall and re-install a port's package by using

```
$ make reinstall
===>  Cleaning for rsnapshot-1.2.9
/usr/sbin/pkg_delete rsnapshot-1.2.9
rsnapshot-1.2.9: complete
Clean shared items: complete
===>  Installing rsnapshot-1.2.9 from /usr/ports/packages/i386/all/rsnapshot-1.2.9.tgz
rsnapshot-1.2.9: complete
```

If you would like to get rid of the packages you just built, you can do so as follows:

```
$ make clean=packages
===>  Cleaning for rsnapshot-1.2.9
rm -f /usr/ports/packages/i386/all/rsnapshot-1.2.9.tgz
```

15.3.8 Using flavors and subpackages

Please do read the `ports(7)` manual page, which gives a good overview of this topic. There are two mechanisms to control the packaging of software according to different needs.

The first mechanism is called **flavors**. A flavor usually indicates a certain set of compilation options. For instance, some applications have a "no_x11" flavor which can be used on systems without X. Some shells have a "static" flavor, which will build a statically linked version. There are ports which have different flavors for building them with different graphical toolkits. Other examples include: support for different database formats, different networking options (SSL, IPv6, ...), different paper sizes, etc.

Summary: Most likely you will use flavors when a package has not been made available for the flavor you are looking for. In this case you will specify the desired flavor and build the port yourself.

Most port flavors have their own working directory during building and every flavor will be packaged into a correspondingly named package to avoid any confusion. To see the different flavors of a certain port, you would change to its subdirectory and issue

```
$ make show=FLAVORS
```

You should also look at the port's DESCR files, as they're supposed to explain the available flavors.

The second mechanism is called **subpackages**. A porter may decide to create subpackages for different pieces of the same application, if they can be logically separated. You will often see subpackages for the client part and the server part of a program. Sometimes extensive documentation is bundled in a separate subpackage because it takes up quite some disk space. Extra functionality that pulls in heavy dependencies will often be packaged separately. The porter will also decide which subpackage is the main subpackage, to be installed as a default. Other examples are: extensive test suites which come with the software, separate modules with support for different things, etc.

Summary: Some ports are split into several packages. `make install` will only install the main subpackage.

To list the different packages built by a port, use

```
$ make show=PKGNAMES
```

`make install` will only install the main subpackage. To install them all, use

```
$ make install-all
```

To list the different subpackages available for a port, use

```
$ make show=MULTI_PACKAGES
```

It is possible to select which subpackage(s) to install from within the ports tree. After some tests, this procedure will just call `pkg_add(1)` to install the desired subpackage(s).

```
$ env SUBPACKAGE="-server" make install
```

Note: The subpackages mechanism only handles packages, it does not modify any configuration options before building the port. For that purpose you must use flavors.

15.3.9 Using dpb to build multiple ports

When you need to build more than one or two ports at a time, you can use a tool provided in /usr/ports/infrastructure/bin - dpb(1). dpb(1) takes a list of ports to build and automatically builds them all in an optimal order and making use of as much parallelism as possible. It can also use multiple machines to perform the building. It produces detailed logs of the build process for troubleshooting, in /usr/ports/log by default.

```
/usr/ports/infrastructure/bin/dpb -P ~/localports
```

will read the list of pkgpaths (pkgpath(7)) in /localports and build all the packages. It can also install the packages after they have been built. /localports would look something like:

```
net/cvsync
www/mozilla-firefox
net/rsync
net/nfsen
textproc/mupdf
misc/magicpoint
lang/sbcl
editors/emacs23
```

If you do not provide a list of ports to build on the command line or via -P or -I, dpb(1) will build all the ports in the ports tree.

15.3.10 Security updates

When serious bugs or security flaws are discovered in third party software, they are fixed in the *-stable* branch of the ports tree. Remember that the lifecycle is 1 release: only the current and last release are updated, as explained in Chapter 5 - OpenBSD's Flavors.

This means all you need to do is make sure you check out the correct branch of the ports tree, and build the desired software from it. You can keep your tree up-to-date with CVS, and in addition subscribe to the ports-changes mailing list to receive security announcements related to software in the ports tree.

Of course, security updates reach the -current ports tree before being taken up in the *-stable* branch.

15.3.11 Package signatures

Signatures are a good way to make sure packages are legitimate and not corrupted.

To start building signed packages, first we need to create a root CA (certificate authority) certificate and key. For the example, we'll use a validity of 10 years.

```
# openssl req -x509 -days 3650 -newkey rsa:2048 -keyout /etc/ssl/private/pkgca.key -out /etc/ssl/
```

Now we are going to create a *build* certificate and key which will be used to sign our packages. For the example, we'll use a validity of 1 year. We will also create a corresponding Certificate Signing Request which will be used by our CA to sign the certificate.

```
# openssl genrsa -out /etc/ssl/private/pkg.key 2048
# openssl req -new -key /etc/ssl/private/pkg.key -out /etc/ssl/private/pkg.csr
```

Now let's sign the certificate using the CA we created in the first step.

```
# openssl x509 -req -days 365 -in /etc/ssl/private/pkg.csr -CA /etc/ssl/pkgca.pem -CAkey /etc/ssl
# rm /etc/ssl/private/pkg.csr
```

Finally, we add the following line to `/etc/mk.conf` to build signed packages by default.

```
PKG_CREATE=/usr/sbin/pkg_create -s x509 -s /etc/ssl/pkg.crt -s /etc/ssl/private/pkg.key
```

When installing signed packages, you will see an added line at the end of the output informing you of the number of signed package(s) you just installed.

```
$ sudo pkg_add vte-0.24.3.tgz
vte-0.24.3: ok
Packages with signatures: 1
```

If you run into trouble dealing with signed packages (e.g. expired certificate...), you can force the (re-)installation and/or removal using one of the following (according to what you want to achieve):

```
$ sudo pkg_add -r -D installed PKGNAME
$ sudo pkg_add -D nosig PKGNAME
$ sudo pkg_delete -q PKGNAME
```

15.4 FAQ

15.4.1 I'm getting all kinds of crazy errors. I just can't seem to get this ports stuff working at all.

It is very likely that you are using a system and ports tree which are not in sync.

Sorry?

- Read EVERYTHING about OpenBSD's Flavors: -release, -stable, and -current. The short summary is as follows, but please do read the document mentioned above to get an idea about which one it is you want to use.

  Release: What is on the CD.

  Stable: Release, plus security and reliability enhancements.

  Current: The development version of OpenBSD.

- Do NOT check out a -current ports tree and expect it to work on a -release or -stable system. This is one of the most common errors and you will irritate people when you ask for help about why "nothing seems to work!" If you follow -current, you need both a -current system and a -current ports tree. Yes, this really does mean a wonderful new port will typically not work on your "older" system – even if that system was -current just a few weeks ago. Keep in mind that if you use X11 as part of your system, it must also follow the corresponding branch!

- Because no intrusive changes are made in -stable, it is possible to use a -stable ports tree on a -release system, and vice versa. There is no need to update all your installed packages after applying a few errata patches to your system.

Another common failure is a missing X11 installation. Even if the port you try to build has no direct dependency on X11, a subpackage of it or its dependencies may require X11 headers and libraries. Building ports on systems without X11 is not supported, so if you insist on doing so, you are on your own to figure it out. For many ports, there are, however, "no_x11" flavored packages available, which you can install without needing X11 on your system.

15.4.2 The latest version of my Top-Favorite-Software is not available!

If you are using a release or stable version of OpenBSD, you will not find any package updates until the next release, or until security issues occur which justify an update of the port in the -stable branch, and of the corresponding package.

WARNING: DO NOT mix versions of Ports and OpenBSD!

Doing so will sooner or later (probably very soon, in fact) cause you headaches trying to solve all kinds of errors!

But hey, I am all -current here!

The ports collection is a volunteer project. Sometimes the project simply doesn't have the developer resources to keep everything up-to-date. Developers pretty much pick up what they consider interesting and can test in their environment.

Some individual ports may lag behind the mainstream versions because of this. Importantly, not updating to the latest version is often a conscious decision. The newer version may have problems that the maintainer is trying to solve, or that have simply made the application worse than the old version. Also, OpenBSD may have different goals than the developers in other projects, which sometimes results in features and design or implementation choices that are undesirable from OpenBSD developers' point of view. The update may also be postponed because the new version is not considered a crucial update.

If you really need a newer version of a port, you should ask the maintainer of the port to update it (see below on how to find out who the port maintainer is). If you can help, all the better.

15.4.3 Why is there no package for my Top-Favorite-Software?

There are several possible reasons for this:

- On the OpenBSD CD-ROMs, there is no space to include every possible package for every possible platform. Therefore only the most used packages are included on the CDs. Additionally, some software can only be redistributed for free, this means it cannot be included on the CDs. If you cannot find a package on the CDs, try another source, such as an FTP mirror.

- Some software must simply not be redistributed in binary package form at all, according to its license. Other software is encumbered by patents and can therefore not be redistributed. If your Top-Favorite-Software falls into this category, you will need to use the port and compile from source.

- Obvious, but sometimes forgotten: there is no port of your Top-Favorite-Software. You can verify this by searching the ports tree. If there is indeed no port of your Top-Favorite-Software, then you are welcome to help.

15.4.4 Why is there no port of my Top-Favorite-Software?

The ports collection is a volunteer project. Active port development is done by a limited number of people, in their spare time. These people usually make new ports only for software they use directly or are interested in.

You can help. Consider creating your own port. There is some documentation available on this: the OpenBSD Porter's Handbook. Read it, and read it again. Especially the part about maintaining your port. Then try making a new port, and test it carefully and step by step. If finally it works OK for you, submit it to the ports mailing list at ports@openbsd.org. Chances are good you will get some feedback and testing from other people. If the testing is successful, your port will be considered to be taken up in the ports tree.

15.4.5 Why is my Top-Favorite-Software not part of the base system?

Because OpenBSD is supposed to be a small stand-alone UNIX-like operating system, we need to draw a line as to what to include. Generally, for an application to be included in the base system:

- It must meet the high quality standards, laid out in the goals of the OpenBSD project.
- Its license must not be too restrictive and must be compatible with the BSD license.
- It must not be too large, in order to keep the size of the base system acceptable.

Further answers to this question are also found in chapter 1.

15.4.6 What should I use: packages or ports?

In general, you are **highly advised** to use packages over building an application from ports. The OpenBSD ports team considers packages to be the goal of their porting work, not the ports themselves.

Building a complex application from source is not trivial. Not only must the application be compiled, but the tools used to build it must be built as well. Unfortunately, OpenBSD, the tools, and the application are all evolving, and often, getting all the pieces working together is a challenge. Once everything works, a revision in any of the pieces the next day could render it broken. Every six months, as a new release of OpenBSD is made, an effort is made to test the building of every port on every platform, but during the development cycle it is likely that some ports will break.

In addition to having all the pieces work together, there is just the matter of time and resources required to compile some applications from source. Applications such as Mozilla products or KDE may take hours and huge amounts

of disk space and RAM/swap to build. Why go through this much time and effort, when the programs are already compiled and sitting on your CD-ROM or FTP mirror, waiting to be used?

Of course, there are a few good reasons to use ports over packages in some cases:

```
Distribution rules prohibit OpenBSD from distributing a package.
You wish to modify or debug the application or study its source code.
You need a flavor of a port that is not built by the OpenBSD ports team.
You wish to alter the directory layout (i.e. modifying PREFIX or SYSCONFDIR).
```

However, for most people and most applications, using packages is a much easier, and definitely the recommended way of adding applications to an OpenBSD system.

15.4.7 How do I tweak these ports to have maximum performance?

OpenBSD is about stability and security. Just like the GENERIC kernel is the default and the only supported kernel, the ports team makes sure the ports work and are stable. If you want to switch on all kinds of compiler options, you are on your own. Please do not ask questions on the mailing lists such as why it does not work, when you tried to switch on a few hidden knobs to make it work faster. In general, all this tweaking is not necessary for more than 99% of users, and it is very likely to be a complete waste of time, for you, the user, as well as for the developers who read about your "problems" when in reality there are none.

15.4.8 I submitted a new port/an update weeks ago. Why isn't it committed?

The ports team has very limited resources and no committer was able to look at your port/update in time. As frustrating as it may be, just ignore this fact. Take care of your port, send updates and eventually someone might take care of it. It has happened before that people suddenly have some free time to spend on committing ports or their interests shift to new areas and suddenly your old submission becomes interesting, if it is remembered.

15.5 Reporting problems

If you have trouble with an existing port, please send e-mail to the port maintainer. To see who is the maintainer of the port, type, for example:

```
$ cd /usr/ports/archivers/unzip
$ make show=MAINTAINER
```

Alternatively, if there is no maintainer, or you can't reach him/her, send e-mail to the OpenBSD ports mailing list, ports@openbsd.org. Please do NOT use the misc@openbsd.org mailing list for questions about ports.

In any case please provide:

- Your OpenBSD version including any patches you may have applied. The kernel version is given by: `sysctl -n kern.version`
- The version of your ports tree: if the file `/usr/ports/CVS/Tag` exists, provide its contents. If this file is absent, you are using the -current ports tree.
- A complete description of the problem. Don't be afraid to provide details. Mention all the steps you followed before the problem occurred. Is the problem reproducible? The more information you provide, the more likely you will get help.

For ports which do not build correctly, a complete build transcript is almost always required. You can use the portslogger script, found in `/usr/ports/infrastructure/bin`, for this. A sample run of portslogger might be:

```
$ mkdir ~/portslogs
$ cd /usr/ports/archivers/unzip
$ make clean install 2>&1 | /usr/ports/infrastructure/bin/portslogger             ~/por
```

After this, you should have a logfile of the build in your /portslogs directory that you can send to the port maintainer. Also, make sure you are not using any special options in your build, for example in `/etc/mk.conf`.

Alternatively, you can

- Use `script(1)` to create a complete build transcript. Do not remove the configure information.
- Attach the output of `pkg_info(1)` if it seems even remotely relevant.
- `gcc(1)` internal compiler errors ask you to report the bug to the gcc mailinglist. It does save time if you follow their direction, and provide at least the various files produced by `gcc -save-temps`.

15.6 Helping us

There are many ways you can help. They are listed below, by increasing order of difficulty.

- Report problems as you experience them.

- You can systematically test ports and report breakages, or suggest improvements. Just have a look at the Port Testing Guide.
- Test the updates to ports which are posted to the ports mailing list.
- Send updates or patches to a port's maintainer, or to the ports mailing list if the port has no maintainer. Generally this is highly appreciated, unless your patches will cause developers to waste time rather than save time.
- Create new ports. If you are really eager and want to know everything about porting applications to OpenBSD, a good starting point is the OpenBSD Porter's Handbook.

Hardware donations can assist testing ports on the various platforms OpenBSD runs on.

Note: For all creation of new ports and subsequent testing, or for testing port updates, you **must run a -current system!** In general, this is not a desirable environment because of its continuously evolving nature, so proceed only if you are sure about committing yourself to following -current.