

Language: [en][\[cs\]](#) [\[de\]](#) [\[es\]](#) [\[ja\]](#) [\[ko\]](#)

Other Documents

[Upgrade-MiniFAQ](#)[Ports and Packages](#)[Port Testing Guide](#)[Updating via AnonCVS](#)[Stable](#)[CVSup](#)[Manual pages](#)[Bug Reporting](#)[Mail lists](#)

PDF files

[FAQ in PDF form](#)

Text files

[FAQ in Text form](#)

Back to OpenBSD



OpenBSD

Documentation and Frequently Asked Questions

[Commonly Encountered Issues](#)[Recent updates](#)

This FAQ is supplemental documentation to the man pages, available both in the installed system and [online](#). The FAQ covers the active release of OpenBSD, currently v3.2. Note that the development version (-current) of OpenBSD is *not* covered by this FAQ.

The FAQ is available in PDF and plain text form in the `pub/OpenBSD/doc` directory from the [FTP mirrors](#).

1 - Introduction to OpenBSD

- [1.1 - What is OpenBSD?](#)
- [1.2 - On what systems does OpenBSD run?](#)
- [1.3 - Is OpenBSD really free?](#)
- [1.4 - Why might I want to use OpenBSD?](#)
- [1.5 - How can I help support OpenBSD?](#)
- [1.6 - Who maintains OpenBSD?](#)
- [1.7 - When will be the next release of OpenBSD?](#)

2 - Other OpenBSD Information Resources

- [2.1 - Web Pages](#)
- [2.2 - Mailing Lists](#)
- [2.3 - Manual Pages](#)
- [2.4 - Reporting Bugs](#)

3 - Obtaining OpenBSD

- [3.1 - Buying an OpenBSD CD](#)
- [3.2 - Buying OpenBSD T-Shirts](#)
- [3.3 - Does OpenBSD provide an ISO image for download?](#)
- [3.4 - Downloading via FTP or AFS](#)
- [3.5 - Obtaining Current Source Code](#)

4 - OpenBSD 3.2 Installation Guide

- [4.1 - Overview of the OpenBSD Installation Procedure.](#)
- [4.2 - Preinstallation Checklist](#)
- [4.3 - Doing an Install](#)
- [4.4 - What files are needed for Installation?](#)
- [4.5 - How much space do I need for an OpenBSD installation?](#)
- [4.6 - Multibooting OpenBSD](#)
- [4.7 - Sending your dmesg to dmesg@openbsd.org after the install](#)
- [4.8 - Adding a file set after install](#)
- [4.9 - What is 'bsd.rd'?](#)
- [4.10 - Common Installation Problems](#)
- [4.11 - Customizing the Install Process](#)
- [4.12 - How can I load a number of similar systems?](#)
- [4.13 - How can I get a dmesg\(8\) to report an install problem?](#)

[5 - Building the System from Source](#)

- [5.1 - OpenBSD Flavors](#)
- [5.2 - Why do I need a custom kernel?](#)
- [5.3 - Kernel configuration Options](#)
- [5.4 - Building your own kernel](#)
- [5.5 - Boot-time configuration](#)
- [5.6 - Getting more verbose output during boot](#)
- [5.7 - Using config\(8\) to change your kernel binary](#)

[6 - Networking](#)

- [6.0.1 - Before we go any further](#)
- [6.1 - Initial network setup](#)
- [6.2 - Packet Filter \(PF\)](#)
- [6.3 - Network Address Translation](#)
- [6.4 - Dynamic Host Configuration Protocol](#)
- [6.5 - Point to Point Protocol](#)
- [6.6 - Tuning networking parameters](#)
- [6.7 - Using NFS](#)
- [6.8 - Domain Name Service - DNS, BIND, and named](#)
- [6.9 - Setting up a PPTP connection in OpenBSD](#)
- [6.10 - Setting up a bridge with OpenBSD](#)

[7 - Keyboard and Display Controls](#)

- [7.1 - How do I remap the keyboard? \(wscons\)](#)
- [7.2 - Is there gpm or the like in OpenBSD?](#)
- [7.3 - How do I clear the console each time a user logs out?](#)
- [7.4 - Accessing the console scrollbar buffer. \(alpha/macppc/i386\)](#)
- [7.5 - How do I switch consoles? \(i386\)](#)
- [7.6 - How can I use a console resolution of 80x50? \(i386\)](#)
- [7.7 - How do I use a serial console?](#)
- [7.8 - How do I blank my console? \(wscons\)](#)

8 - General Questions

- [8.2 - How do I change virtual terminals? \(i386 ONLY\)](#)
- [8.3 - I forgot my root password... What do I do!](#)
- [8.4 - X won't start, I get lots of error messages](#)
- [8.5 - What is CVS, and how do I use it?](#)
- [8.6 - What is the ports tree?](#)
- [8.7 - What are packages?](#)
- [8.8 - Is there any way to use my floppy drive if it's not attached during boot?](#)
- [8.9 - OpenBSD Bootloader \(i386 specific\)](#)
- [8.10 - Using S/Key on your OpenBSD system](#)
- [8.11 - Why is my Macintosh losing so much time?](#)
- [8.12 - Does OpenBSD support SMP?](#)
- [8.13 - I sometimes get Input/output error when trying to use my tty devices](#)
- [8.14 - What web browsers are available for OpenBSD?](#)
- [8.15 - How do I use the mg editor?](#)
- [8.16 - Ksh does not appear to read my .profile!](#)
- [8.17 - Why does my /etc/motd file get written over when I modified it?](#)
- [8.18 - Why does www.openbsd.org run on Solaris?](#)
- [8.19 - I'm having problems with PCI devices being detected](#)
- [8.20 - Antialiased and TrueType fonts in OpenBSD 2.9/XFree86](#)
- [8.21 - Does OpenBSD support any journaling filesystems?](#)
- [8.22 - Reverse DNS or Why is it taking so long for me to log in?](#)
- [8.23 - Why do the OpenBSD web pages not conform to HTML4/XHTML?](#)
- [8.24 - Why is my clock off by twenty-some seconds?](#)

9 - Migrating from Linux

- [9.1 - Tips for Linux \(and other free Unix-like OS\) users](#)
- [9.2 - Dual boot of Linux and OpenBSD](#)
- [9.3 - Converting your Linux \(or other System-7 style\) password file to BSD-style.](#)
- [9.4 - Getting OpenBSD and Linux to interact](#)

10 - System Management

- [10.1 - When I try to su to root it says that I'm in the wrong group](#)
- [10.2 - How do I duplicate a filesystem?](#)
- [10.3 - How do I start daemons with the system? \(Overview of rc\(8\)\)](#)
- [10.4 - Why do users get relaying access denied when they are remotely sending mail through my OpenBSD system?](#)
- [10.5 - I've set up POP, but I get errors when accessing my mail through POP. What can i do?](#)
- [10.6 - Why does Sendmail ignore /etc/hosts file?](#)
- [10.7 - Setting up a Secure HTTP Server using SSL\(8\)](#)
- [10.8 - I made changes to /etc/passwd with vi\(1\), but the changes didn't seem to take place. Why?](#)
- [10.9 - How do I add a user? or delete a user?](#)

- [10.10 - How do I create a ftp-only account?](#)
- [10.11 - Setting up user disk quotas](#)
- [10.12 - Setting up Kerberos Client/Server](#)
- [10.13 - Setting up an Anonymous FTP Server](#)
- [10.14 - Confining users to their home dir's in ftpd\(8\).](#)
- [10.15 - Applying patches in OpenBSD.](#)
- [10.16 - Tell me about chroot\(\) Apache?](#)
- [10.17 - I don't like the standard root shell!](#)
- [10.18 - What else can I do with ksh?](#)

11 - Performance Tuning

- [11.1 - Networking](#)
- [11.2 - Disk I/O](#)
- [11.4 - Hardware Choices](#)
- [11.5 - Why aren't we using async mounts?](#)
- [11.6 - Tuning your monitor resolution under XFree86](#)

12 - For Advanced Users

- [12.1 - Forcing DMA access for IDE disks](#)
- [12.2 - Upgrading from various versions of OpenBSD via CVS.](#)

13 - Using IPsec (IP Security Protocol)

- [13.1 - What is IPsec?](#)
- [13.2 - That's nice, but why do I want to use IPsec?](#)
- [13.3 - What are the protocols behind IPsec?](#)
- [13.4 - On the wire format](#)
- [13.5 - Configuring IPsec](#)
- [13.6 - How do I set up IPsec with manual keying?](#)
- [13.7 - How do I set up isakmpd?](#)
- [13.8 - How do I use isakmpd with X.509 certificates?](#)
- [13.9 - What IKE clients are compatible with isakmpd?](#)
- [13.10 - Troubleshooting IPsec/VPN](#)
- [13.11 - Related Documentation](#)

14 - Disk Setup

- [14.1 - Using OpenBSD's disklabel](#)
- [14.2 - Using OpenBSD's fdisk](#)
- [14.3 - Adding extra disks in OpenBSD](#)
- [14.4 - How to swap to a file](#)
- [14.5 - Soft Updates](#)
- [14.6 - When I boot after installation of OpenBSD/i386, it stops at "Using Drive: 0 Partition 3".](#)
- [14.7 - What are the issues regarding large drives with OpenBSD? -i386 specific](#)
- [14.8 - Installing Bootblocks - i386 specific](#)

- [14.9 - Preparing for disaster: Backing up and Restoring from tape.](#)
 - [14.10 - Mounting disk images in OpenBSD](#)
 - [14.11 - Help! I'm getting errors with PCIIDE!](#)
 - [14.12 - Forcing DMA access for IDE disks](#)
 - [14.13 - RAID options with OpenBSD](#)
-

Commonly Encountered Issues

- [Tell me about chroot\(\) Apache?](#)
 - [How do I upgrade my system?](#)
 - [How do I update my system?](#) and [here](#)
 - [RAID Options.](#)
 - [Packet Filter](#) and [NAT.](#)
 - [How do I set up a multi-boot system?](#)
 - [Issues with Large Drives and OpenBSD](#)
 - [How do I blank my console?](#)
-

Recent Updates

- [FAQ 4, How can I get a dmesg\(8\) to report an install problem?](#)
 - [FAQ 4 Customizing the Install Process](#)
 - [FAQ 4, How can I load a number of similar systems?](#)
 - [FAQ 10, I don't like the standard root shell!](#)
 - [FAQ 10, What else can I do with ksh?](#)
 - [FAQ 4, Common Installation Problems](#)
 - [FAQ 7, How do I blank my console?](#)
-

The FAQ maintainers are:

Nick Holland, Eric Jackson, Wim Vandeputte and Chris Cappuccio.

For information about and assisting in the translation of this FAQ and the rest of the OpenBSD website, see the [translation page](#).

Questions and comments regarding the FAQ may be directed to faq@openbsd.org.


General questions about OpenBSD should be directed to the appropriate [mail list](#).

Back to OpenBSD



OpenBSD FAQ Copyright © 1998-2003 OpenBSD

\$OpenBSD: index.html,v 1.166 2003/03/17 19:22:51 nick Exp \$



"If you don't find it in the index, look very carefully through the entire catalogue."

Sears, Roebuck, and Co., Consumer's Guide, 1897



[\[FAQ Index\]](#) [\[To Section 2 - Other OpenBSD Information Resources\]](#)

1 - Introduction to OpenBSD

Table of Contents

- [1.1 - What is OpenBSD?](#)
 - [1.2 - On what systems does OpenBSD run?](#)
 - [1.3 - Is OpenBSD really free?](#)
 - [1.4 - Why might I want to use OpenBSD?](#)
 - [1.5 - How can I help support OpenBSD?](#)
 - [1.6 - Who maintains OpenBSD?](#)
 - [1.7 - When will be the next release of 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](#). OpenBSD supports binary emulation of most binaries from SVR4 (Solaris), FreeBSD, Linux, BSDI, SunOS, and HP-UX.

1.2 - On what systems does OpenBSD run?

OpenBSD 3.2 runs on the following platforms:

- [i386](#) - CD bootable
- [sparc](#) - CD bootable
- [sparc64](#) - CD bootable
- [hp300](#) - CD bootable
- [amiga](#)
- [mac68k](#)
- [macppc](#) - CD bootable
- [mvme68k](#)
- [alpha](#) - CD bootable
- [vax](#)

bootable means that OpenBSD will boot directly from the CD. The CD set will boot on several hardware platforms. See [chapter 3](#) of this FAQ for details of obtaining OpenBSD on CD.

Previous releases of OpenBSD also had a port for:

- [sun3](#) - This port was removed after the 2.9 release
- [arc](#) - This port was removed after the 2.3 release
- [mvme88k](#)
- [pmax](#)

If you are interested in multiprocessor support, see [FAQ 8, Multiprocessor](#) for more info.

1.3 - 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.

For further reading on other popular licenses read: <http://www.openbsd.org/policy.html>.

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.

1.4 - 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 un-answerable 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. Whether OpenBSD is right for you is a question that only you can answer.

- OpenBSD runs on many [different hardware platforms](#).
- OpenBSD is thought of by many security professionals to be the most [secure](#) UNIX-like operating system as the result of a 10-member 1.5-year long 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 offers opportunities for ordinary people to participate in the development and testing of the product.

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 and SCSI 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 will be the next release of OpenBSD?

The OpenBSD team makes a new release every six months, with target release dates of May 1 and November 1. More information on the development cycle can be found [here](#).

[\[FAQ Index\]](#) [\[To Section 2 - Other OpenBSD Information Resources\]](#)



www@openbsd.org

\$OpenBSD: faq1.html,v 1.46 2003/04/04 15:44:53 nick Exp \$



[\[FAQ Index\]](#) [\[To Section 1 - Introduction to OpenBSD\]](#) [\[To Section 3 - Obtaining OpenBSD\]](#)

2 - Other OpenBSD Information Resources

Table of Contents

- [2.1 - Web Pages](#)
 - [2.2 - Mailing Lists](#)
 - [2.3 - Manual Pages](#)
 - [2.4 - Reporting Bugs](#)
-

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.

Additional information for laptop users can be found at:
<http://www.monkey.org/openbsd-mobile/>.

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. The confirmation you send back to the list processor will be included in its reply to you. You get an option of a hypertext link or sending a reply to the message to confirm your intent. An email reply should contain something like this:

```
accept A56D-70D4-52C3
```

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 instructions. They can be obtained by sending an e-mail message to majordomo@openbsd.org with a message body of "help". Some of the most popular OpenBSD mailing lists are:

- **announce** - Important announcements. This is a low-volume list.
- **security-announce** - Announcements of security issues.
- **misc** - General user questions and answers. This is the most active list, and should be the "default" for most questions.
- **tech** - Technical topics for OpenBSD developers and advanced users. Please direct 'new user' and installation-related questions to *misc*, not *tech*. Please do not cross-post to *misc* and *tech*.
- **bugs** - Bugs received via sendbug(1) and discussions about them.
- **source-changes** - Automated mailing of CVS source tree changes.
- **ports** - Discussion of the OpenBSD Ports Tree.
- **ports-changes** - Automated mailing of ports-specific CVS source tree changes.
- **advocacy** - Discussion on advocating OpenBSD.

Before posting a question on **misc** or any other mail 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 mail lists may have seen the same question several times in the last week, and may not appreciate seeing it again. You can find several archives and other mail list guidelines on the mailing lists web page: <http://www.openbsd.org/mail.html>.

Another mailing list that may be of interest is openbsd-mobile@monkey.org. This mailing list is a discussion of the use of OpenBSD in mobile computing.

To subscribe to this list use:

```
'echo subscribe | mail "openbsd-mobile-request@monkey.org"'
```

The archives for that can be found at: <http://www.monkey.org/openbsd-mobile/archive/>.

2.3 - Manual Pages

OpenBSD comes with extensive documentation in the form of manual pages, as well as longer documents relating to specific applications. To access the manual pages and other documentation, be sure that you installed the man and misc sets.

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

- [help\(1\)](#) - help for new users and administrators.
- [afterboot\(8\)](#) - things to check after the first complete boot.
- [boot\(8\)](#) - system bootstrapping procedures.
- [login.conf\(5\)](#) - format of the login class configuration file.
- [adduser\(8\)](#) - command for adding new users.
- [vipw\(8\)](#) - edit the master password file.
- [man\(1\)](#) - display the on-line manual pages.
- [sendbug\(1\)](#) - send a problem report (PR) about OpenBSD to a central support site.
- [disklabel\(8\)](#) - Read and write disk pack label.
- [ifconfig\(8\)](#) - configure network interface parameters.
- [route\(8\)](#) - manually manipulate the routing tables.
- [netstat\(1\)](#) - show network status.
- [reboot, halt\(8\)](#) - Stop and restart the system.
- [shutdown\(8\)](#) - close down the system at a given time.
- [boot_config\(8\)](#) - how to change kernel configuration at boot.

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 man32.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) - daemon to execute scheduled commands (Vixie Cron)
crontab (1) - maintain crontab files for individual users (V3)
crontab (5) - tables for driving cron
# man 5 crontab
```

In addition to the UNIX manual pages, there is a typesettable document set (included in the misc distribution). It lives in the `/usr/share/doc` directory. If you also installed the text distribution, then you can format each document set with a ``make'` in the appropriate subdirectory. The `psd` subdirectory is the Programmer's Supplementary Documents distribution. The `smm` subdirectory is the System Manager's Manual. The `usd` subdirectory is the UNIX User's Supplementary Documents distribution. You can perform your ``make'` in the three distribution subdirectories, or you can select a specific section of a distribution and do a ``make'` in its subdirectory. Some of the subdirectories are empty. By default, formatting the documents will result in Postscript output, suitable for printing. The Postscript output can be quite large -- you should assume 250-300% increase in volume. If you do not have access to a Postscript printer or display, you may also format the documents for reading on a terminal display. In each Makefile you'll need to add the flag `-Tascii` to each instance of the [groff\(1\)](#) commands (or execute it by hand). Some of the documents use the `ms` formatting macros, and some use the `me` macros. The Makefile in each document subdirectory (eg, `/usr/share/doc/usd/04.csh/Makefile`) will tell you which one to use. For example:

```
# cd /usr/share/doc/usd/04.csh
# groff -Tascii -ms tabs csh.1 csh.2 csh.3 csh.4 csh.a csh.g > csh.txt
# more csh.txt
```

The UNIX manual pages are generally more current and trustworthy than the typesettable documents. The typesettable documents sometimes explain complicated applications in more detail than the manual pages do.

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`).

This is 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 :

```
# nroff -mdoc <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 `[man_src_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 [Ports Tree](#). Use the following [groff\(1\)](#) command options for getting a PostScript version from an OpenBSD system man page:

```
# groff -mdoc -Tps [man_src_file] > outfile.ps
```

The above command line will only work for man pages formatted with the [mdoc\(7\)](#) macro package, used to format the BSD man pages. For getting a PostScript version from a third party software man page (either self-compiled or installed from the [ports\(7\)](#) or the [packages\(7\)](#)), do instead:

```
# groff -Tps -mandoc [man_src_file] > outfile.ps
```

2.4 - Reporting Bugs

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:

(Note: See [report.html](#) for more information on creating and submitting bug reports. Basically, detailed information about your hardware is necessary if you think the bug is in any way related to your hardware or hardware configuration. Usually, [dmesg\(8\)](#) output is sufficient in this respect. Next, a detailed description of your problem is necessary.)

```
From: smartuser@example.com
To: bugs@openbsd.org
Subject: 2.7 panics on an i386
```

```
After installing OpenBSD 2.7 from the CD which I purchased via your outstanding on-line ordering system, I find that the system halts when using any network utilities. After booting with a bootdisk and escaping to a shell. This is the dmesg output:
```

```
OpenBSD 2.7 (GENERIC) #690: Fri Oct 29 16:32:17 MDT 1999
  deraadt@i386.openbsd.org:/usr/src/sys/arch/i386/compile/GENERIC
cpu0: F00F bug workaround installed
cpu0: Intel Pentium (P54C) ("GenuineIntel" 586-class) 120 MHz
cpu0: FPU,V86,DE,PSE,TSC,MSR,MCE,CX8
BIOS mem  = 654336 conventional, 15728640 extended
real mem  = 16384000
avail mem = 11112448
using 225 buffers containing 921600 bytes of memory
mainbus0 (root)
bios0 at mainbus0: AT/286+(63) BIOS, date 08/20/96
bios0: diskinfo 0xe055800c cksumlen 1 memmap 0xe0558088 apminfo 0xe0558134
apm0 at bios0: Power Management spec V1.1
apm0: battery life expectancy 95%
apm0: AC on, battery charge high, charging, estimated 1:27 minutes
pci0 at mainbus0 bus 0: configuration mode 1 (no bios)
pchb0 at pci0 dev 0 function 0 "Toshiba (2nd ID) Host-PCI" rev 0x11
"Chips and Technologies 65550" rev 0x04 at pci0 dev 4 function 0 not configured
isa0 at mainbus0
isadma0 at isa0
wdc0 at isa0 port 0x1f0/8 irq 14
wd0 at wdc0 channel 0 drive 0: <TOSHIBA MK2720FC>
wd0: can use 16-bit, PIO mode 4
wd0: 16-sector PIO, LBA, 1296MB, 2633 cyl, 16 head, 63 sec, 2654280 sectors
sb0 at isa0 port 0x220/24 irq 5 drq 1: dsp v3.02
midi0 at sb0: <SB MIDI UART>
audio0 at sb0
```

```
opl0 at sb0: model OPL3
midi1 at opl0: <SB Yamaha OPL3>
wss0 at isa0 port 0x530/8 irq 10 drq 0: CS4232 (vers 63)
audiol at wss0
pcppi0 at isa0 port 0x61
midi2 at pcppi0: <PC speaker>
sysbeep0 at pcppi0
npx0 at isa0 port 0xf0/16: using exception 16
pccom0 at isa0 port 0x3f8/8 irq 4: ns16550a, 16 byte fifo
pccom1 at isa0 port 0x2f8/8 irq 3: ns16550a, 16 byte fifo
pccom2: irq 5 already in use
vt0 at isa0 port 0x60/16 irq 1: generic VGA, 80 col, color, 8 scr, mf2-kbd
pms0 at vt0 irq 12
fdc0 at isa0 port 0x3f0/6 irq 6 drq 2
fd0 at fdc0 drive 0: 1.44MB 80 cyl, 2 head, 18 sec
pcic0 at isa0 port 0x3e0/2 iomem 0xd0000/16384
pcic0 controller 0: <Intel 82365SL rev 1> has sockets A and B
pcmcia0 at pcic0 controller 0 socket 0
pcmcial at pcic0 controller 0 socket 1
ne3 at pcmcial function 0 "Linksys, EtherFast 10/100 PC Card (PCMPC100), " port 0x340/16 irq 9
ne3: address 00:e0:98:04:95:ba
pcic0: irq 11
biomask 4040 netmask 4240 ttymask 5a42
pctr: 586-class performance counters and user-level cycle counter enabled
dkcsum: wd0 matched BIOS disk 80
root on wd0a
rootdev=0x0 rrootdev=0x300 rawdev=0x302
```

Thank you!

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.

After submitting a bug report, you will be notified by E-mail about the status of the report. 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 mail list, details on the [mail list page](#), or query the bug report database status at the on-line [Bug Tracking System](#).

[\[FAQ Index\]](#) [\[To Section 1 - Introduction to OpenBSD\]](#) [\[To Section 3 - Obtaining OpenBSD\]](#)



www@openbsd.org

\$OpenBSD: faq2.html,v 1.58 2003/03/18 03:41:04 nick Exp \$



[\[FAQ Index\]](#) [\[To Section 2 - Other OpenBSD Information Resources\]](#) [\[To Section 4 - Installation Guide\]](#)

3 - Obtaining OpenBSD

Table of Contents

- [3.1 - Buying an OpenBSD CD](#)
 - [3.2 - Buying OpenBSD T-Shirts](#)
 - [3.3 - Does OpenBSD provide an ISO image for download?](#)
 - [3.4 - Downloading via FTP or AFS](#)
 - [3.5 - Obtaining Current Source Code](#)
-

3.1 - Buying an OpenBSD CD

Purchasing an OpenBSD CD is generally the best way to get started. Visit the ordering page to purchase your copy: <http://www.openbsd.org/orders.html>.

There are many good reasons to own an OpenBSD CD:

- 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 purchase has a real impact on future development.
- The CD contains binaries (and source) for all supported platforms.
- The CD is bootable on several platforms, and can be used to bootstrap a machine without a pre-existing installed operating system.
- The CD is 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.

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

3.2 - Buying OpenBSD T-Shirts

Yes, OpenBSD has t-shirts for your wearing enjoyment. You can view these at <http://www.OpenBSD.org/tshirts.html>. Enjoy :)

3.3 - Does OpenBSD provide an ISO image for download?

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, helping 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. If for some reason you want to download a CD image, try searching the mailing list archives for possible sources. Of course, any OpenBSD ISO images available on the Internet either violate Theo de Raadt's copyright or are not official images. The source of an unofficial image may or may not be trustworthy; it is up to you to determine this for yourself.

We suggest that people who want to download OpenBSD for free use the FTP install option.

3.4 - Downloading via FTP or AFS

There are numerous international mirror sites offering FTP access to OpenBSD releases and snapshots. AFS access is also available. You should always use the site closest 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 nearest mirror is performing adequately. Of course, your OpenBSD release CD is always closer than any mirror. Access information is here:

<http://www.openbsd.org/ftp.html>.

3.5 - Obtaining Current Source Code

Source to OpenBSD is freely redistributable and available at no charge. Generally the best way to get started with a current source tree is to install the source from the most recent CD and then configure AnonCVS to update it regularly. Information about AnonCVS, including how to set it up, is available here:

<http://www.openbsd.org/anoncv.html>.

or see [FAQ 8, CVS](#)

If you don't have sufficient network bandwidth to support AnonCVS, or if your Internet access is via UUCP, you can still keep your source current by using CTM instead of AnonCVS. If that's your situation, then starting with a recent release CD is even more important. Information about CTM, including how to set it up, is available here:

<http://www.openbsd.org/ctm.html>.

Yet another alternative is to get the source code from the web. You can do that through cvsweb at: <http://www.openbsd.org/cgi-bin/cvsweb/>.

[\[FAQ Index\]](#) [\[To Section 2 - Other OpenBSD Information Resources\]](#) [\[To Section 4 - Installation Guide\]](#)



www@openbsd.org

\$OpenBSD: faq3.html,v 1.37 2003/03/18 03:41:04 nick Exp \$



[\[FAQ Index\]](#) [\[To Section 3 - Obtaining OpenBSD\]](#) [\[To Section 5 - Building the System from Source\]](#)

4 - OpenBSD 3.2 Installation Guide

Table of Contents

- [4.1 - Overview of the OpenBSD Installation Procedure](#)
 - [4.1.1 - Supported OpenBSD Architectures](#)
 - [4.1.2 - Supported Installation Media](#)
 - [4.1.3 - Creating bootable OpenBSD install floppies](#)
 - [4.1.3.1 - Creating Floppies on Unix](#)
 - [4.1.3.2 - Creating Floppies on DOS/Windows](#)
 - [4.1.4 - Booting OpenBSD Installation Images](#)
 - [4.2 - Preinstallation Checklist](#)
 - [4.3 - Doing an install](#)
 - [4.3.1 - Setting up Disks](#)
 - [4.3.2 - Setting the System Hostname](#)
 - [4.3.3 - Configuring the Network](#)
 - [4.3.4 - Choosing Installation Media](#)
 - [4.3.5 - Choosing Filesets](#)
 - [4.3.6 - Small RAM procedure](#)
 - [4.3.7 - Finishing up](#)
 - [4.3.8 - Other Information Resources](#)
 - [4.4 - What files are needed for Installation?](#)
 - [4.5 - How much space do I need for an OpenBSD installation?](#)
 - [4.6 - Multibooting OpenBSD](#)
 - [4.7 - Sending your dmesg to dmesg@openbsd.org after the install](#)
 - [4.8 - Adding a file set after install](#)
 - [4.9 - What is 'bsd.rd'?](#)
 - [4.10 - Common Installation Problems](#)
 - [4.10.1 - Install hangs during MAKEDEV](#)
 - [4.10.2 - My Compaq only recognizes 16M RAM](#)
 - [4.10.3 - My i386 won't boot after install](#)
 - [4.10.4 - My machine booted, but hung at the ssh-keygen process](#)
 - [4.10.5 - I got the message "ftplist: No such file" when doing an install](#)
 - [4.11 - Customizing the Install Process](#)
 - [4.12 - How can I load a number of similar systems?](#)
 - [4.13 - How can I get a dmesg\(8\) to report an install problem?](#)
-

4.1 - Overview of the OpenBSD installation procedure

OpenBSD has a very robust and adaptable text-based install procedure. In addition to its robustness, the install procedure can be done using 1 floppy disk. Most architectures have a similar installation procedure; however there are some differences in details. In all cases, you are urged to read the platform-specific INSTALL document in the platform directory on the CDROM or FTP sites (for example, *INSTALL.i386*, *INSTALL.mac68k* or *INSTALL.sparc*).

On most architectures, you have several installation options, including FTP, CDROM, and local disk files. One option **not** supported is [downloading an ISO image](#) to make your own official CDROM. CDROMs are available for [purchase](#), however.

4.1.1 - Supported OpenBSD Architectures

OpenBSD 3.2 supports a number of architectures listed below in alphabetical order. Please refer to each architecture's page for specific information on what each architecture supports.

- [alpha](#) - DEC Alpha-based machines.
- [amiga](#) - Amiga m68k-based models (MMU required).
- [hp300](#) - Hewlett-Packard HP300/HP400 machines.
- [i386](#) - Intel i386 compatibles.
- [mac68k](#) - Most MC680x0-based Apple Macintosh models.
- [mvme68k](#) - Motorola MVME147/16x/17x 68K VME cards.
- [macppc](#) - Support for Apple based PowerPC systems.
- [sparc](#) - SPARC Platform by Sun Microsystems.
- [sparc64](#) - Sun's UltraSPARC systems.
- [vax](#) - DEC's VAX computers.

4.1.2 - Supported Installation Media

OpenBSD can be installed from multiple media types. The most common and architecture independent options are laid out below. These options can be used after booting from an OpenBSD CD-ROM, floppy disk or [ramdisk kernel](#).

CD-ROM	To do a CD-ROM install, you must have either purchased an Official OpenBSD CD-ROM or created your own OpenBSD CD. This is usually the easiest way to install an OpenBSD system. NOTE: Official OpenBSD CDs are bootable if your BIOS supports it.
FTP	This installation option allows you to install OpenBSD by downloading the installation packages in realtime over the network.
Local Filesystem	This option allows you to install from files on a pre-existing filesystem. Support for DOS, EXT2FS and FFS are included on the i386 install disk.

4.1.3 - Creating bootable OpenBSD install floppies.

To create an installation floppy image you must either download the correct boot floppy image from one of the OpenBSD distribution sites or copy the image from an Official OpenBSD CD-ROM. You can find a list of FTP servers at the [OpenBSD FTP Distribution](#) page. Most architectures have one or more boot floppy images to choose from; different images are for different hardware variations. The differences between the i386 platform installation floppies will be outlined below. For the other architectures with multiple boot floppies, see the INSTALL document in the respective FTP directory. For those with only one, just download the respective *floppy32.fs* image.

NOTE: The *cdrom32.fs* image can be used to make a bootable OpenBSD installation CD-ROM.

The [i386](#) platform has four separate installation disk images to choose from:

- *floppy32.fs* (Desktop PC) supports many PCI and ISA NICs, IDE and simple SCSI adapters and some PCMCIA support.
- *floppyB32.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.
- *floppyC32.fs* (Laptops) supports the Cardbus and PCMCIA devices found in many laptops.
- *cdrom32.fs* it is, in effect a combination of all three boot disks. It can be used to make a bootable 2.88M floppy, or more commonly, as a boot image for a recordable CD.

Most i386 users will just use the *floppy32.fs* installation floppy. Yes, there may be situations where one install disk is required to support your SCSI adapter and another disk is required to support your network adapter. Fortunately, this is a rare event, and can usually be worked around.

Once you have the correct floppy image, you need to get a clean floppy disk. If there are ANY bad sectors on the floppy disk, the installation will most likely fail.

4.1.3.1 - Creating floppies on Unix.

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

```
# fdformat /dev/fd0a
Format 1440K floppy `/dev/fd0a'? (y/n): y
processing VVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVV done.
```

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

one.

Once you have a clean, formatted floppy it is time to write the installation image to floppy. For this, you can use the [dd\(1\)](#) utility. An example usage of *dd(1)* is below:

```
# dd if=floppy32.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 floppy32.fs
```

4.1.3.2 - Creating floppies on DOS/Windows.

If you are creating this image on the Windows/DOS platform you can get tools mentioned below from the [tools](#) directory on any of the ftp mirrors, or in *3.2/tools* directory on CD1 of the OpenBSD CD set. For users of DOS/Windows 9x, *rawrite* can be used to write your boot floppy.

To prepare a floppy in MS-DOS or Windows, simply use the native formatting tools.

To write the installation image to the prepared floppy you can use *rawrite*, *fdimage*, or *ntrw*. *rawrite* will not work on Windows NT, 2000 or XP.

Note that *FDIMAGE.EXE* and *RAWRITE.EXE* are both MS-DOS applications, and thus, are limited to DOS's "8.3" file naming convention. As *floppy32B.fs* and *floppy32C.fs* have longer file names, you will have to find out how your system stored the file in "8.3 format" before using *FDIMAGE.EXE* or *RAWRITE.EXE* to make your boot floppies.

Example usage of *rawrite*:

```
C:\> rawrite
RaWrite 1.2 - Write disk file to raw floppy diskette

Enter source file name: floppy32.fs
Enter destination drive: a
Please insert a formatted diskette into drive A: and press -ENTER- : Enter
```

Example Usage of *fdimage*:

```
C:\> fdimage -q floppy32.fs a:
```

Example Usage of *ntrw*:

```
C:\> ntrw floppy32.fs a:
3.5", 1.44MB, 512 bytes/sector
bufsize is 9216
1474560 bytes written
```

4.1.4 - Booting OpenBSD Installation Images.

This section is initially broken down into architecture dependent sections for popular architectures that OpenBSD supports. This is so we can properly instruct each user on what to do on their respective platform.

Booting i386

Booting an install image on the i386 architecture is nothing new to most people. If you are using the floppy disk, simply stick the floppy into your floppy drive and boot your system. Your install image will automatically load (assuming floppy boot is enabled in your BIOS). If you are planning on booting from CD, you must go into your systems BIOS and set the boot options to allow booting from CD. Some older BIOSs 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.

Booting sparc

NOTE: On the sparc64, only the sbus machines (Ultra 1, Ultra 2) are bootable from floppy.

To boot from floppy, place your floppy disk with the OpenBSD installation image on it into your floppy drive. Then use the following command to boot from your floppy:

```
ok boot floppy
```

To boot from CD-ROM, place your OpenBSD CD-ROM disk into your drive. If your Sun only has one CD-ROM drive, then just go to the boot prompt, where you can 'boot cdrom':

```
ok boot cdrom
```

Of course, this will only work in new command mode. If you are at the old command mode prompt (a right arrow), type 'n' for the new command mode. (If you are using an old sparc that is pre-sun4c, you probably don't have a new command mode. In this case, you need to experiment.) If you have multiple CD-ROM devices, you need to boot from the correct one. Try `probe-scsi` from the new command mode.

```
ok probe-scsi

Target 0
  Unit 0   Disk      QUANTUM LIGHTNING 365S
Target 1
  Unit 0   Removable Disk  QUANTUM EMPIRE_1080S
Target 3
  Unit 0   Removable Disk  Joe's CD ROMs
```

Figure out which disk is the CD ROM you want to boot from. Note the target number.

```
ok boot /sbus/esp/sd@X,0
```

4.2 - Preinstallation 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 (CDROM, FTP, etc.)
- Desired disk layout
 - Does existing data need to be saved elsewhere?
 - Will OpenBSD co-exist 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 will an existing (or future) partition have to be worked around?
 - 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 servers)
 - IP addresses and subnet masks for each NIC
 - Gateway address
- Will you be running X on this system?

4.3 - 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. The most common problem when booting is a bad floppy disk or a drive alignment problem. The boot floppy is quite tightly packed -- any bad spot will cause problems.

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 where. Don't worry about remembering this text, as a copy is saved as `/var/run/dmesg.boot`. On most architectures, `SHIFT+PGUP` will let you examine text that has scrolled off the screen.

Then, you will see the following:

```
rootdev=0x1100 rrootdev=0x2f00 rawdev=0x2f02
erase ^?, werase ^W, kill ^U, intr ^C, status ^T
(I)nstall, (U)pgrade or (S)hell? i
```

And with that, we reach our first question. Most of the time, 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. You will not be given the option of installing the `etc32.tgz` file. After the install, you will have to manually merge the changes of `etc32.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.
NOTE: The Upgrade process is not designed to skip releases! While this will often work, it is not supported. For OpenBSD 3.2, upgrading 3.1 to 3.2 is the only supported upgrade. If you have to upgrade from an older version, a complete reinstall is recommended.
- **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.

On occasion, you will not see the "Upgrade" option listed. After a [flag day](#) event, it is not possible to directly upgrade; one must rebuild the system from scratch. An example of this in OpenBSD 3.2 is the Sparc platform, which switched executable formats from `a.out` to `ELF`, and so a user upgrading to 3.2 on their Sparc must completely reload their system.

In this example, we will do an install, but the upgrade process is similar.

```
Welcome to the OpenBSD/i386 3.2 install program.
```

```
This program will help you install OpenBSD in a simple and rational way. At
any prompt except password prompts you can run a shell command by typing
'!foo', or escape to a shell by typing '!'. Default answers are shown in []'s
and are selected by pressing RETURN. At any time you can exit this program by
pressing Control-C and then RETURN, but quitting during an install can leave
your system in an inconsistent state.
```

```
Specify terminal type: [vt220]
Do you wish to select a keyboard encoding table? [n] y
```

In most cases, the default terminal type is appropriate; however if you are using a serial console for install, don't just take the default, respond appropriately.

If you do not select a keyboard encoding table, a US keyboard layout will be assumed.

```
IS YOUR DATA BACKED UP? As with anything that modifies disk contents, this
program can cause SIGNIFICANT data loss.
```

```
It is often helpful to have the installation notes handy. For complex disk
configurations, relevant disk hardware manuals and a calculator are useful.
```

```
Proceed with install? [n] y
```

If you take the default here, the install process will terminate and drop you to a shell prompt.

4.3.1 - Setting up disks during installation.

Setting up disks in OpenBSD varies a bit between platforms. For [i386](#) and [macppc](#), 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 several 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 can coexist with other Operating Systems on the IBM AT descended machines. The second layer of partitioning is visible only to OpenBSD and those OSs which can directly read an OpenBSD filesystem.

```
Cool! Let's get to it...
```

```
You will now initialize the disk(s) that OpenBSD will use. To enable all
available security features you should configure the disk(s) to allow the
creation of separate filesystems for /, /tmp, /var, /usr, and /home.
```

```
Available disks are: wd0.
Which one is the root disk? (or done) [wd0] Enter
```

The root disk is the disk the system will boot from, and normally where swap space resides. Usually, this will be the default -- if it isn't, you will need to know how to force your computer to boot from a non-standard disk. IDE disks will show up as `wd0`, `wd1`, etc., SCSI disks and RAID devices will show up as `sd0`, `sd1`, and so on. All the disks OpenBSD can find are listed here -- if you have drives which are not showing up, you have unsupported or improperly configured hardware.

```
Do you want to use *all* of wd0 for OpenBSD? [no] Enter
```

If you say "yes" to this question, the entire disk will be allocated to OpenBSD. This will result in a simple Master Boot Record and partition table being written out to disk -- one partition, the size of the entire hard disk, set to the OpenBSD partition type, and flagged as the bootable partition. This will be a common option for most production uses of OpenBSD; however, there are some systems this should not be done on. Many Compaq systems, some Dell and other systems use a "maintenance" partition, which should be kept intact. If your system has any other partitions of any type you do not wish to erase, do not select "yes" to the above question.

For the sake of this example, we will assume the disk is to be split between OpenBSD and a pre-existing Windows 2000 partition, so we take the default of "no", which will take us into the [fdisk\(8\)](#) program. You can also get more information on `fdisk(8)` [here](#).

Important Note: Users with a large hard disk (larger than 8G on a newer i386, though on older machines and different platforms, often much smaller) will want to see [this section](#) before going any further.

```
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.
```

```
The 'manual' command describes all the fdisk commands in detail.
```

```
Disk: wd0          geometry: 2586/240/63 [39100320 Sectors]
Offset: 0         Signature: 0xAA55

#  id  C  H  S  -  C  H  S  [  LBA Info:  start:  size  ]
-----
*0: 06  0  1  1  -  202 239 63 [ 63: 3069297 ] 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
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
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>

```

A few commands are worthy of elaboration:

- **r** or **reinit**: Clears existing partition table, makes one big OpenBSD partition, flags it active. Equivalent to saying "yes" to the "use *all* of ..." question.
- **p** or **print**: Displays the current partition table in sectors. "p m" will show the partition table in megabytes, "p g" will show it in gigabytes.
- **e** or **edit**: edit or alter a table entry.
- **f** or **flag**: Marks a partition as the active partition, the one that will be booted from
- **exit** and **quit**: Careful on these, as some users are used to "exit" and "quit" having opposite meanings.

It is worth pointing out once again, a screwup here will result in significant data loss. If you are going to do this on a drive with important data, it might be worth practicing on a "disposable" drive, in addition to having a good backup.

Our drive here has a 1.5G partition for Windows 2000 (using the FAT filesystem). Looking at the info from the above display, we can see that the Windows partition occupies through cylinder 202 on the drive. So, we are going to allocate the rest of the disk to OpenBSD, starting at cylinder 203. You could also calculate OpenBSD's starting sector of 3069360 by adding the existing partition's starting sector (63) and its size (3069297).

You can edit the drive layout in either Cylinder/Heads/Sectors form or just raw sectors. Which is easier depends upon what you are doing; in this case, working around an existing partition, using CHS format will probably be easier.

```

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] y
BIOS Starting cylinder [0 - 2585]: [0] 203
BIOS Starting head [0 - 239]: [0] Enter
BIOS Starting sector [1 - 63]: [0] 1
BIOS Ending cylinder [0 - 2585]: [0] 2585
BIOS Ending head [0 - 239]: [0] 239
BIOS Ending sector [1 - 63]: [0] 63
fdisk:*1> p
Disk: wd0      geometry: 2586/240/63 [39100320 Sectors]
Offset: 0      Signature: 0xAA55
      Starting      Ending      LBA Info:
  #: id   C  H  S -   C  H  S [   start:      size   ]
-----
*0: 06   0  1  1 -  202 239 63 [         63:    3069297 ] DOS > 32MB
  1: A6  203  0  1 - 2585 239 63 [    3069360:   36030960 ] 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> p m
Disk: wd0      geometry: 2586/240/63 [19092 Megabytes]
Offset: 0      Signature: 0xAA55
      Starting      Ending      LBA Info:
  #: id   C  H  S -   C  H  S [   start:      size   ]
-----
*0: 06   0  1  1 -  202 239 63 [         63:    1499M] DOS > 32MB
  1: A6  203  0  1 - 2585 239 63 [    3069360:   17593M] OpenBSD
  2: 00   0  0  0 -   0  0  0 [         0:         0M] unused
  3: 00   0  0  0 -   0  0  0 [         0:         0M] unused
fdisk:*1>

```

Note that the prompt changed to include an asterisk (*) to indicate you have unsaved changes. As we can see from the output of `partition` we have not altered our Windows partition, we have successfully allocated the rest of the drive for OpenBSD, and the partitions do not overlap. We are in business. Almost.

What we haven't done is flagged the partition as active so the machine will boot OpenBSD on the next reboot:

```
fdisk:*1> f 1
Partition 1 marked active.
fdisk:*1> p
Disk: wd0          geometry: 2586/240/63 [39100320 Sectors]
Offset: 0          Signature: 0xAA55
-----
#  id  C  H  S  -  C  H  S  [  LBA Info:  start:  size  ]
-----
0: 06  0  1  1  - 202 239 63 [          63:  3069297 ] DOS > 32MB
*1: A6 203 0  1  - 2585 239 63 [ 3069360: 36030960 ] 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>
```

And now, we are ready to save our changes:

```
fdisk:*1> w
Writing MBR at offset 0.
wd0: no disk label
fdisk: 1> q
```

Creating a disklabel

The next step is to use [disklabel\(8\)](#) to slice up the OpenBSD partition. More details on using `disklabel(8)` can be found in [FAQ 14, disklabel](#).

Here is the partition information you chose:

```
Disk: wd0          geometry: 2586/240/63 [39100320 Sectors]
Offset: 0          Signature: 0xAA55
-----
#  id  C  H  S  -  C  H  S  [  LBA Info:  start:  size  ]
-----
*0: 06  0  1  1  - 202 239 63 [          63:  3069297 ] DOS > 32MB
1: A6 203 0  1  - 2585 239 63 [ 3069360: 36030960 ] OpenBSD
2: 00  0  0  0  -   0  0  0 [          0:         0 ] unused
3: 00  0  0  0  -   0  0  0 [          0:         0 ] unused
```

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.

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.

```
disklabel: no disk label
WARNING: Disk wd0 has no label. You will be creating a new one.

# using MBR partition 1: type A6 off 3069360 (0x2ed5b0) size 36030960 (0x225c9f0)

Treating sectors 3069360-39100320 as the OpenBSD portion of the disk.
You can use the 'b' command to change this.

Initial label editor (enter '?' for help at any prompt)
> ?
Available commands:
  p [unit] - print label.
  M        - show entire OpenBSD man page for disklabel.
  e        - edit drive parameters.
  a [part] - add new partition.
```



```

b          - set OpenBSD disk boundaries.
c [part]  - change partition size.
d [part]  - delete partition.
D          - set label to default.
g [d|b]   - Use [d]isk or [b]ios geometry.
m [part]  - modify existing partition.
n [part]  - set the mount point for a partition.
r          - recalculate free space.
u          - undo last change.
s [path]  - save label to file.
w          - write label to disk.
q          - quit and save changes.
x          - exit without saving changes.
X          - toggle expert mode.
z          - zero out partition table.
? [cmdnd] - this message or command specific help.
Numeric parameters may use suffixes to indicate units:
'b' for bytes, 'c' for cylinders, 'k' for kilobytes, 'm' for megabytes,
'g' for gigabytes or no suffix for sectors (usually 512 bytes).
Non-sector units will be rounded to the nearest cylinder.
Entering '?' at most prompts will give you (simple) context sensitive help.
>

```

Again, a few of these commands could use a little elaboration:

- **p** - displays (prints) the current disklabel to the screen, and you can use the modifiers **k**, **m** or **g** for kilobytes, megabytes or gigabytes.
- **D** - Clears any existing disklabel, creates a new default disklabel which covers just the current OpenBSD partition. This can be useful if the disk previously had a disklabel on it, and the OpenBSD partition was recreated to a different size -- the old disk label may not get deleted, and may cause confusion.
- **m** - Modifies an existing entry in a disklabel. Do not over estimate what this will do for you. While it may alter the size of a disklabel partition, it will NOT alter the filesystem on the drive. Using this option and expecting it to resize existing partitions is a good way of losing large amounts of data.

Slicing up your disk properly is important. The answer to the question, "How should I partition my system?" is "Exactly how you need it". This will vary from application to application. There is no universal answer. If you are unsure of how you want to partition your system, see [this discussion](#).

In this system, we have over 17G available for OpenBSD. That's a lot of space, and it isn't likely we will need most of it. So, we will deliberately not use absolute minimum sizes. We would rather have a few hundred megabytes of unused space than a kilobyte too little.

On the root disk, the two partitions 'a' and 'b' **must** be created. The installation process will not proceed until these two partitions are available. 'a' will be used for the root filesystem (/) and 'b' will be used as swap space.

After a little thought, we decide to create just enough partitions to allow the creation of the recommended separate filesystems (/ , /tmp, /var, /usr, /home) along with a swap partition:

- **wd0a:** / (root) - 150M. Should be more than enough.
- **wd0b:** (swap) - 300M.
- **wd0d:** /tmp - 120M. /tmp is used for building some software, 120M will probably be enough for most things.
- **wd0e:** /var - 80M. If this were to be a web or mail server, we'd have made this partition much larger, but, that's not what we are doing.
- **wd0g:** /usr - 2G. We want this partition to be large enough to load quite a few user applications, plus be able to update and rebuild the system if desired or needed. The [Ports tree](#) will be here as well, which will take almost 100M of this space before ports are built.
- **wd0h:** /home - 4G. This will allow plenty of user file space.

Now, if you add those up, you will see over 10G of space is unused! Unused space won't hurt anything, and it gives us flexibility to enlarge things in the future if need be. Need more /tmp? No problem, create a new one in the unused space, change /etc/fstab and problem solved.

```

> p m
device: /dev/rwd0c
type: ESDI
disk: ESDI/IDE disk
label: ST320011A
bytes/sector: 512
sectors/track: 63
tracks/cylinder: 16
sectors/cylinder: 1008
cylinders: 16383
total sectors: 39102336
free sectors: 36030960
rpm: 3600

16 partitions:
#      size  offset  fstype  [fsize bsize  cpgr]
  a: 17593.2M 1498.7M  unused          0    0
  c: 19092.9M   0.0M  unused          0    0
  i: 1498.7M   0.0M  MSDOS

> d a
> a a
offset: [3069360] Enter
size: [36030960] 150M
Rounding to nearest cylinder: 307440
FS type: [4.2BSD] Enter
mount point: [none] /
> a b
offset: [3376800] Enter
size: [35723520] 300M
Rounding to nearest cylinder: 614880
FS type: [swap] Enter
> a d
offset: [3991680] Enter
size: [35108640] 120m
Rounding to nearest cylinder: 245952
FS type: [4.2BSD] Enter
mount point: [none] /tmp
> a e
offset: [4237632] Enter
size: [34862688] 80m
Rounding to nearest cylinder: 164304
FS type: [4.2BSD] Enter
mount point: [none] /var
> a g
offset: [4401936] Enter
size: [34698384] 2g
Rounding to nearest cylinder: 4194288
FS type: [4.2BSD] Enter
mount point: [none] /usr
> a h
offset: [8596224] Enter
size: [30504096] 4g
Rounding to nearest cylinder: 8388576
FS type: [4.2BSD] Enter
mount point: [none] /home
> p m
device: /dev/rwd0c
type: ESDI
disk: ESDI/IDE disk
label: ST320011A
bytes/sector: 512
sectors/track: 63
tracks/cylinder: 16
sectors/cylinder: 1008
cylinders: 16383
total sectors: 39102336
free sectors: 22115520
rpm: 3600

16 partitions:
#      size  offset  fstype  [fsize bsize  cpgr]
  a: 150.1M 1498.7M 4.2BSD 1024 8192 16 # /
  b: 300.2M 1648.8M  swap

```

```

c: 19092.9M    0.0M    unused      0    0
d:  120.1M   1949.1M   4.2BSD     1024  8192    16 # /tmp
e:   80.2M   2069.2M   4.2BSD     1024  8192    16 # /var
g:  2048.0M   2149.4M   4.2BSD     1024  8192    16 # /usr
h:  4096.0M   4197.4M   4.2BSD     1024  8192    16 # /home
i:  1498.7M    0.0M     MSDOS
> q
Write new label?: [y] Enter

```

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. You will also note the *i* partition wasn't defined by us; this is the pre-existing Windows 2000 partition. Partitions are not assigned any particular letters -- with the exception of *a* (root), *b* (swap) and *c* (entire disk), the rest of the partitions (through letter *p*) are available for use as you desire.

If you look closely at the output of the `disklabel`, you will note that your drive RPM rating is probably wrong. This is historical; the drive speed is not used in any way by the system. Do not worry about it.

Configuring your mount points and formatting your filesystems

Now comes the final configuration of your mount points. If you configured the mount points through [disklabel\(8\)](#), this step consists of just verifying your selections; otherwise, you can specify them now.

```

The root filesystem will be mounted on wd0a.
wd0b will be used for swap space.
Mount point for wd0d (size=122976k), none or done? [/tmp] Enter
Mount point for wd0e (size=82152k), none or done? [/var] Enter
Mount point for wd0g (size=2097144k), none or done? [/usr] Enter
Mount point for wd0h (size=4194288k), none or done? [/home] Enter
Mount point for wd0d (size=122976k), none or done? [/tmp] done
Done - no available disks found.

```

You have configured the following partitions and mount points:

```

wd0a /
wd0d /tmp
wd0e /var
wd0g /usr
wd0h /home

```

The next step creates a filesystem on each partition, ERASING existing data.

```

Are you really sure that you're ready to proceed? [n] y
/dev/rwd0a:    307440 sectors in 305 cylinders of 16 tracks, 63 sectors
              150.1MB in 20 cyl groups (16 c/g, 7.88MB/g, 1920 i/g)
/dev/rwd0d:    245952 sectors in 244 cylinders of 16 tracks, 63 sectors
              120.1MB in 16 cyl groups (16 c/g, 7.88MB/g, 1920 i/g)
/dev/rwd0e:    164304 sectors in 163 cylinders of 16 tracks, 63 sectors
              80.2MB in 11 cyl groups (16 c/g, 7.88MB/g, 1920 i/g)
/dev/rwd0g:    4194288 sectors in 4161 cylinders of 16 tracks, 63 sectors
              2048.0MB in 261 cyl groups (16 c/g, 7.88MB/g, 1920 i/g)
/dev/rwd0h:    8388576 sectors in 8322 cylinders of 16 tracks, 63 sectors
              4096.0MB in 521 cyl groups (16 c/g, 7.88MB/g, 1920 i/g)
/dev/wd0a on /mnt type ffs (rw, asynchronous, local, ctime=Thu Oct 10 21:
50:36 2 002)
/dev/wd0h on /mnt/home type ffs (rw, asynchronous, local, nodev, nosuid,
ctime=Thu Oct 10 21:50:36 2002)
/dev/wd0d on /mnt/tmp type ffs (rw, asynchronous, local, nodev, nosuid,
ctime=Thu Oct 10 21:50:36 2002)
/dev/wd0g on /mnt/usr type ffs (rw, asynchronous, local, nodev, ctime=Th
u Oct 10 21:50:36 2002)
/dev/wd0e on /mnt/var type ffs (rw, asynchronous, local, nodev, nosuid,
ctime=Th u Oct 10 21:50:36 2002)

```

You may wonder why the installer again asks for mount points. This allows you to recover from any errors or omissions in the mount points specified during the creation of the `disklabel`. For instance, the installation process will automatically delete any duplicate mount points you enter during the configuration of the `disklabel`. The `disklabel` program will allow you to enter such duplicates, and thus they must be checked for after the `disklabel` program exits. The deleted duplicate mount points will result in partitions without mount points, that you must assign new mount points for if you wish to use the space.

Notice the "Are you really sure that you are ready to proceed?" question defaults to no, so you will have to deliberately tell it to proceed and format your partitions. If you chose no, you would simply be dropped into a shell and could start the install again by typing `install`, or just by rebooting again with your boot disk.

At this point all filesystems will be formatted for you. This could take some time depending on the size of the partitions and the speed of the disk.

4.3.2 - Setting the System Hostname

Now you must set the system hostname. This value will be saved in the file `/etc/myname`, which is used during normal boots to set the hostname of the system. As of OpenBSD 3.2 the name stored in `/etc/myname` includes the fully qualified domain name of the system (FQDN). If you do not set the FQDN of the system, the default value of 'my.domain' will be used.

It is important to set this name now, because it will be used when the cryptographic keys for the system are generated during the first boot after installation. This generation takes place whether the network is configured or not.

```
Enter system hostname (short form, e.g. 'foo'): puffy
```

4.3.3 - Configuring your Network

Now it is time to configure your network. The network must be configured if you are planning on doing a ftp or nfs based install, considering it will be based upon the information you are about to enter. Here is a walk through of the network configuration section of the install process.

```
Configure the network? [y] Enter

If any interface will be configured by DHCP, you should not enter
information that will be supplied via DHCP, e.g. the DNS domain name.

Enter DNS domain name (e.g. 'bar.com'): [my.domain] example.com
Available interfaces are: fxp0.
Which one do you wish to initialize? (or done) [fxp0] Enter
IP address for fxp0 (or 'dhcp')? 199.185.137.55
Symbolic (host) name? [puffy] Enter
Netmask? [255.255.255.0] Enter
The default media for fxp0 is
    media: Ethernet autoselect (100baseTX full-duplex)
Do you want to change the default media? [n] Enter
Done - no available interfaces found.
Enter IP address of default route: [none] 199.185.137.128
Enter IP address of primary nameserver: [none] 199.185.137.1
Would you like to use the nameserver now? [y] Enter
Do you want to do more, manual, network configuration? [n] Enter
```

In the above example, we use a static IP address. You can choose to use dhcp as well if you wish. In the case of DHCP, most of this information will be grabbed from a remote dhcp server, though you will be given a chance to confirm it.

NOTE: Only **one** interface can easily be configured using DHCP during an install. If you attempt to configure more than one interface using DHCP you will encounter errors. You have to manually configure the additional interfaces after the installation.

4.3.4 - Choosing Installation Media

After your network is set up, the install script will give you a chance to make manual adjustments to the configuration. Then the filesystems you created will be mounted and a root password set. This will get your local disks ready for the OpenBSD packages to be installed upon them.

Next, you will get a chance to choose your installation media. The options are listed below.

```
You will now specify the location and names of the install sets you want to
load. You will be able to repeat this step until all of your sets have been
successfully loaded. If you are not sure what sets to install, refer to the
installation notes for details on the contents of each.
```

```
Sets can be located on a (m)ounted filesystem; a (c)drom, (d)isk or (t)ape
device; or a (f)tp, (n)fs or (h)ttp server.
```

```
Where are the install sets you want to use? (m, c, f, etc.) c
Available CD-ROMs are: cd0.
```

In this example we are installing from CD-ROM. This will bring up a list of devices on your computer identified as a CD-ROM. Most people will only have one. If you need to make sure you pick the device which you will use to install OpenBSD from.

NOTE: All possible sources for install sets are listed, but not all may be available on your system. e.g. (n)fs is shown but not all architectures allow NFS installations. If you choose a source that is not available, you will get an error message and be given the chance to choose another source for your installation sets.

```
Available CD-ROMs are: cd0.
Which one contains the install media? (or done) [cd0] Enter

Enter the pathname where the sets are stored (or '?') [3.2/i386] Enter
```

Here, you are prompted for which directory the installation files are, which is `3.2/i386/` on the official CDROM.

4.3.5 - Choosing Filesets.

Now it's time to choose which packages you will be installing. You can get a description of these files in [the next section](#). The files that the install program finds will be shown to you on the screen. Your job is just to specify which files you want. By default all the non-X packages are selected; however, some people may wish to limit this to the bare minimum required to run OpenBSD, which would be `base32.tar.gz`, `etc32.tar.gz` and `bsd`. Others will wish to install all packages. The example below is that of a full install.

```
The following sets are available. Enter a filename, 'all' to select
all the sets, or 'done'. You may de-select a set by prepending a '-'
to its name.
```

```
[X] base32.tgz
[X] etc32.tgz
[X] misc32.tgz
[X] comp32.tgz
[X] man32.tgz
[X] game32.tgz
[ ] xbase32.tgz
[ ] xshare32.tgz
[ ] xfont32.tgz
[ ] xserv32.tgz
[X] bsd
```

```
File Name? (or 'done') [xbase32.tgz] all
```

```
The following sets are available. Enter a filename, 'all' to select
all the sets, or 'done'. You may de-select a set by prepending a '-'
to its name.
```

```
[X] base32.tgz
[X] etc32.tgz
[X] misc32.tgz
[X] comp32.tgz
[X] man32.tgz
[X] game32.tgz
[X] xbase32.tgz
[X] xshare32.tgz
[X] xfont32.tgz
[X] xserv32.tgz
[X] bsd
```

You can do all kinds of nifty things here -- `-x*` would remove all X components, if you changed your mind. In this case, we are going to load all the sets. While the system will run with fewer sets, the starting default is recommended.

Once you have successfully picked which packages you want, you will be prompted to make sure you want to extract these packages and they will then be installed. A progress bar will be shown that will keep you informed on how much time it will take. The times range greatly depending on what system it is you are installing OpenBSD on, the packages installed, and the speed of the source media. This part may from a few minutes to several hours.

```
File Name? (or 'done') [done] Enter
Ready to install sets? [y] Enter

Getting base32.tgz ...
100% |*****| 23870 KB    00:15
Getting etc32.tgz ...
100% |*****| 1447 KB    00:01
Getting misc32.tgz ...
100% |*****| 1666 KB    00:00
Getting comp32.tgz ...
100% |*****| 16801 KB   00:12
Getting man32.tgz ...
100% |*****| 5428 KB    00:04
Getting game32.tgz ...
100% |*****| 2702 KB    00:01
Getting bsd ...
100% |*****| 4409 KB    00:01
Getting xbase32.tgz ...
100% |*****| 8837 KB    00:04
Getting xshare32.tgz ...
100% |*****| 1531 KB    00:02
Getting xfont32.tgz ...
100% |*****| 30664 KB   00:14
Getting xserv32.tgz ...
100% |*****| 14798 KB   00:05
Extract more sets? [n]
```

If your system has a small amount of RAM (less than 20M on i386), do not hit return at that prompt quite yet!

4.3.6 - Special steps for machines with little RAM

As OpenBSD has grown, the minimum RAM requirement has grown, and is likely to continue to increase. The next step in the install process will require more than 16M RAM, and will fail on older machines with less than 20M RAM (again, these are i386 numbers -- some other platforms will require far less RAM, though this trick can be used with them as well, if they are running near the lower-limits).

During the install process, there is normally no swap; the real RAM is all you have. The 'MAKEDEV' step that follows will require more than the rest of the install required. As a system with small amounts of RAM can still be very usable for many applications, working around this limitation is a quite useful trick.

The solution is to activate swap now. The swap partition has been created, the files are installed to the hard disk, the only trick here is to manually invoke it. So, do not just hit 'ENTER' at the above prompt, but rather, hit '!', which will bring up a shell, and launch [swapon\(8\)](#) from the mounted hard drive:

```
Extract more sets? [n] !
Type 'exit' to return to install.
# /mnt/sbin/swapon /dev/wd0b
total: 307440k bytes allocated = 0k used, 307440k available
# exit
Extract more sets? [n] Enter
```

You can now resume the normal installation.

4.3.7 - Finishing up

You will now be asked if you plan to run X on this system. If you answer 'Y', `/etc/sysctl.conf` will be modified to include the line `machdep.allowaperture=1` or `machdep.allowaperture=2`, depending on your platform.

Your last task is to enter the time zone. Depending on where your machine lives, there are may be several equally valid answers for the question. In the example that follows, we used US/Eastern, but could also have used EST5EDT or US/Michigan and had the same result. Hitting ? at the prompts will guide you through your choices.

```

Extract more sets? [n] Enter
Do you expect to run the X Window System? [y] y
Saving configuration files.....done.
Generating initial host.random file .....done.
What timezone are you in? ('?' for list) [US/Pacific] ?
Africa/      Chile/      GB-Eire      Israel       NZ-CHAT      Turkey
America/     Cuba        GMT          Jamaica     Navajo       UCT
Antarctica/  EET         GMT+0        Japan        PRC          US/
Arctic/      EST         GMT-0        Kwajalein   PST8PDT      UTC
Asia/        EST5EDT     GMT0         Libya       Pacific/     Universal
Atlantic/    Egypt       Greenwich    MET          Poland       W-SU
Australia/   Eire        HST          MST          Portugal     WET
Brazil/      Etc/        Hongkong     MST7MDT     ROC          Zulu
CET          Europe/     Iceland      Mexico/     ROK          posix/
CST6CDT     Factory     Indian/      Mideast/    Singapore    posixrules
Canada/      GB          Iran          NZ           SystemV/     right/
What timezone are you in? ('?' for list) [US/Pacific] US
Select a sub-timezone of 'US' ('?' for list): ?
Alaska      Central      Hawaii       Mountain     Samoa
Aleutian    East-Indiana  Indiana-Starke  Pacific
Arizona     Eastern      Michigan     Pacific-New
Select a sub-timezone of 'US' ('?' for list): Eastern
You have selected timezone 'US/Eastern'.

```

The last steps are for the system to create the /dev directory (which may take a while on some systems, especially if you had to use the [Small RAM](#) trick above), and install the boot blocks.

```

Making all device nodes...done.
Installing boot block...
boot: /mnt/boot
proto: /usr/mdec/biosboot
device: /dev/rwd0c
/usr/mdec/biosboot: entry point 0
proto bootblock size 512
room for 12 filesystem blocks at 0x16f
Will load 7 blocks of size 8192 each.
Using disk geometry of 63 sectors and 240 heads.
 0:  9 @(203 150 55) (3078864-3078872)
 1: 63 @(203 151 1) (3078873-3078935)
 2: 24 @(203 152 1) (3078936-3078959)
 3: 16 @(203 8 47) (3069910-3069925)
/mnt/boot: 4 entries total
using MBR partition 1: type 166 (0xa6) offset 3069360 (0x2ed5b0)
...done.

CONGRATULATIONS! Your OpenBSD install has been successfully completed!
To boot the new system, enter halt at the command prompt. Once the
system has halted, reset the machine and boot from the disk.
# halt
syncing disks... done

The operating system has halted.
Please press any key to reboot.

```

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

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 Ports and Packages system for installing software](#), as well as [here](#) and [here](#)

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.

4.3.8 - Other Information Resources

Additional documents already exist for those of you who might have special needs or interests. You can retrieve these from any of the [mirror ftp sites](#).

- [INSTALL.i386](#) - Comprehensive installation document. Similar documents exist for all platforms.
- [INSTALL.linux](#) - Installing OpenBSD along with Linux.
- [INSTALL.mbr](#) - Explaining the Master Boot Record.
- [INSTALL.pt](#) - Explaining Partition Tables.
- [INSTALL.dbr](#) - DOS Floppy Disk Boot Sector.
- [INSTALL.chs](#) - Explaining CHS Translation.
- [INSTALL.ata](#) - ATA/ATA-1/ATA-2/IDE/EIDE/etc FAQ
- [INSTALL.os2br](#) - The os2 Boot Sector.

4.4 - 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. Here is an overview of each:

- *base32.tgz* - Contains the base OpenBSD system **Required**
- *etc32.tgz* - Contains all the files in /etc **Required**
- *comp32.tgz* - Contains the compiler and its tools, libs. **Recommended**
- *man32.tgz* - Contains man pages **Recommended**
- *misc32.tgz* - Contains misc info, setup documentation
- *game32.tgz* - Contains the games for OpenBSD
- *xbase32.tgz* - Contains the base install for X11
- *xfont32.tgz* - Contains X11's font server and fonts
- *xserv32.tgz* - Contains X11's X servers
- *xshare32.tgz* - Contains manpages, locale settings, includes, etc for X
- *bsd* - This is the Kernel. **Required**

4.5 - How much space do I need for an OpenBSD installation?

The following are minimum suggested filesystem sizes for a full system install. The numbers include enough extra space to permit you to run a typical home system that is connected to the Internet.

- These are minimum values.
- If you plan to install a significant amount of third party software, make your /usr partition large! **At least** triple these values!
- For a system that handles lots of email or web pages (stored, respectively, in /var/mail and /var/www) you will want to make your /var partition significantly larger, or put them on separate partitions.
- For a multiuser system which may generate lots of logs, you will still want to make your /var partition significantly larger (/var/log).
- If you plan to rebuild the kernel or system from source, you will want to make the /usr partition significantly larger, **at least** 800M-1G larger than indicated below.

As you read this, keep in mind that `/usr` and `/usr/X11R6` are usually both parts of the same filesystem, that is, `/usr`, as there is no big advantage to making them into separate filesystems.

SYSTEM	/	/usr	/var	/usr/X11R6
alpha	50M	250M	25M	100M
amiga	40M	200M	25M	180M
hp300	40M	200M	25M	80M
i386	40M	200M	25M	140M
mac68k	40M	200M	25M	80M
macppc	50M	200M	25M	100M
mvme68k	40M	200M	25M	80M
sparc	40M	259M	25M	49M
sparc64	50M	200M	25M	100M
vax	75M	125M	25M	180M

In addition, it is recommended that a `/tmp` partition be used. The `/tmp` partition is used in the compiling of ports, among other things, so how big you make it depends on what you do with it. 50M may be plenty for most people, but some large applications may require 100M or more of `/tmp` space.

When you are in the disklabel editor, you may choose to make your entire system have just an 'a' (main filesystem) and 'b' (swap). The 'a' filesystem which you set up in disklabel will become your root partition, which should be the sum of all the 3 main values above (`/`, `/usr`, and `/var`) plus some space for `/tmp`. The 'b' partition you set up automatically becomes your system swap partition -- we recommend a minimum of 32MB but if you have disk to spare make it at least 64MB. If you have lots of disk space to spare, make this 256MB, or even 512MB.

There are five main reasons for using separate filesystems, instead of shoving everything into one or two filesystems:

- **Security:** You can mark some filesystems as 'nosuid', 'nodev', 'noexec', 'readonly', etc. This is now done by the install process, in fact, if you use the above described 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 of course 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, what OpenBSD uses, is not prone to heavy fragmentation.)
- **Integrity:** If one filesystem is corrupted for some reason then your other filesystems are still OK.
- **Size:** Many platforms 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 (8G on new i386 systems). As the kernel can end up anywhere in 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.

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`.
- 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 [upgrade-minifaq](#) 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 20G, 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 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.

4.6 - Multibooting OpenBSD (i386)

Multibooting is having several operating systems on one computer, and some means of selecting the 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 *highly* 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.

When multibooting, the requirements of all operating systems must be met by your configuration. People often ask if there is a way around the [8G boot limit](#) of OpenBSD. While there are some programs that claim to get around various limits of various operating systems, none of them are known to do this with current versions of OpenBSD.

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 a `/etc/boot.conf` file (yes, you will also have to create an `/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.

In this case, 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\(1\)](#):

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

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](#).

Note: The Windows NT 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.

Other boot loaders

Some other bootloaders OpenBSD users have used successfully include [GAG](#), [OSBS](#), [The Ranish Partition Manager](#) and [GRUB](#).

OpenBSD and Linux (i386)

Please refer to [INSTALL.linux](#), which gives in depth instructions on getting OpenBSD working with Linux.

4.7 - 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.

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
    dmesg | mail dmesg@openbsd.org
so that we can see what kinds of configurations people are running. We will
use this information to improve device driver support in future releases.
(We would be much happier if this information was for the supplied GENERIC
kernel; not for a custom compiled kernel). The device driver information
we get from this helps us fix existing drivers.
```

Make sure you send email from an account that is able to also receive email so developers can contact you back 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 | mail your-account@yourmail.dom
```

and then forward that message to

```
dmesg@openbsd.org
```

where your-account@yourmail.dom is your regular email account. (or transfer the dmesg output using ftp/scp/floppydisk/carrier-pigeon/...)

NOTE - Please send only GENERIC kernel dmesgs. Custom kernels that have device drivers removed are not helpful.

4.8 - 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 *comp32.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 xzvpf comp32.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 a missing file sets. This does not work. `pkg_add(1)` is for package files, not generic tar files like the install sets.

4.9 - 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 1.29
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.

4.10 - Common Installation Problems

4.10.1 - Install hangs during MAKEDEV

Usually, this is caused by a lack of physical RAM in the system. See [here](#).

4.10.2 - 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/CDROM's boot> prompt, load the system, reboot, and create a */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.

4.10.3 - 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 answer "Y" to the "Use entire disk for OpenBSD?" 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. The solution is to boot the install media again, drop to the shell and invoke [fdisk\(8\)](#) and use the "update" option.
- **In some rare occasions, something may go wrong with the second stage boot loader install.** Reinstalling the second stage boot loader is discussed [here](#).

4.10.4 - 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 45 minutes or more to complete the three [ssh-keygen\(1\)](#) steps, some machines will take even longer. Just let it finish; it is only done once per install.

4.10.5 - I got the message "ftplist: No such file" 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:

```
Do you want to see a list of potential FTP servers? [y] ENTER
grep: /tmp/ftplist: No such file or 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 CDROM.

4.11 - 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. 32). 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 `xzpf`. This set is intended to be installed last, after all other file sets (*-current* install/upgrade scripts guarantee this, though 3.2-release does not. Manually select this file set after the primary installation has been completed).

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.

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](#), 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, 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 almost full functionality of your system (though, in single user mode) when your script runs.

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 anything possible in a script.

- 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.
- Do an [immediate backup/archive](#) of the new system before you expose it to the rest of the world.

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

4.12 - How can I load 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

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.

4.13 - 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 CDROM has 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:** See [this article](#) on setting up the serial console, then capture the output to a file.
- **FTP:** Under some circumstances, you may be able to use the [ftp\(1\)](#) client on the boot disk or CDROM to send the dmesg to a local FTP server, where you can retrieve it later.

[\[FAQ Index\]](#) [\[To Section 3 - Obtaining OpenBSD\]](#) [\[To Section 5 - Building the System from Source\]](#)



www@openbsd.org

\$OpenBSD: faq4.html,v 1.137 2003/04/04 15:44:53 nick Exp \$

OpenBSD

[\[FAQ Index\]](#) [\[To Section 4 - Installation Guide\]](#) [\[To Section 6 - Networking\]](#)

5 - Building the System from Source

Table of Contents

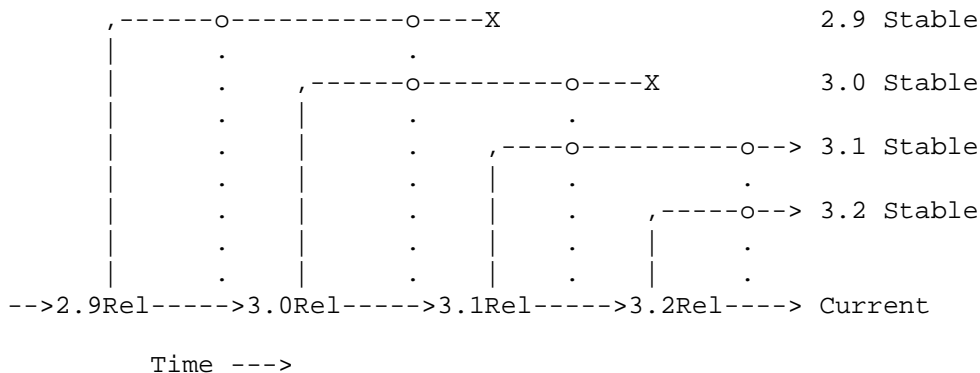
- [5.1 - OpenBSD's Flavors](#)
 - [5.2 - Why do I need a custom kernel?](#)
 - [5.3 - Kernel configuration Options](#)
 - [5.4 - Building your own kernel](#)
 - [5.5 - Boot-time configuration](#)
 - [5.6 - Getting more verbose output during boot](#)
 - [5.7 - Using config\(8\) to change your kernel binary](#)
-

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:** The to-the-moment version of OpenBSD, which will turn into the next release.

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



The code tree is branched between *-current*, *-release* and *-stable* every six months -- *-release* becomes a frozen point (a "Tag") in the history of the source tree - it is never changed, and is what is on the [CDs](#) and [FTP servers](#). *-Current* is where active work is done, and turns into the next *-release* of OpenBSD.

The *-stable* (also known as the "Patch branch") is based on *-release*, and as one can see, it is a "branch" from the development path of OpenBSD. As important fixes are made to *-current*, they are "back ported" into the *-stable* branches. In the above illustration, the vertical dotted lines are bug fixes being incorporated into the *-stable* branches. You will also note that in the above example, the 2.9-*stable* branch came to an end with 3.1-*release*, and the 3.0-*stable* branch came to an end with 3.2-*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 3.2-*release* to 3.2-*stable*).

The *-stable* branch is *-release* plus patches found on the [errata page](#), and some simple fixes that do not merit an errata entry. Usually, 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 will NOT be added to *-stable*, and new feature support will rarely be added unless it is considered very important.

Warning: *-current* is a moving target. It changes almost minute by minute, and may well change several times in the time it takes to retrieve the source code. As indicated earlier, there is no promise that the system compiles or that it works (of course, the hope is it does). It is entirely possible and not uncommon to get the *-current* source and have it fail to compile, whereas five minutes later, it may work just fine. **If you are not prepared to deal with this, stay away from *-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 some 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 where "normal" users may wish to live on the cutting edge and run *-current*. The most common reason is when the user has a device which is not supported by *-release* (and thus, not *-stable*), or wishes to use a new feature of the *-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, this 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 usually a good starting point before attempting to build *-current*.

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, one might be able to build several snapshots in a 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.

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 with a *-release* userland. Yes, this does mean you need to update 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 when [upgrading by source](#), the update process is supported in **only one direction: from older to**

newer, and from *-stable* to *-current*. You can not run *3.2-current* (or a snapshot), then decide you are living too dangerously, and step back to *3.2-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 long and hard before committing yourself to using *-current*.

5.2 - Why would I want to create my own custom kernel?

Several reasons, although this practice is generally geared towards knowledgeable users who have a good overall understanding of the system.

- Your computer has a very small amount of RAM and you want to preserve as much as possible by removing device drivers you don't use
- You wish to remove default options or add options which may not have been enabled by default
- You wish to enable experimental options

Under most circumstances you will NOT need to compile your own kernel. The GENERIC kernel will usually be all that you need. In fact, there are several reasons why you do not want to create your own kernel. The main reason is that it is very easy to make changes to the kernel configuration which look logical, but do not work. This is your danger sign. If something does not appear to work properly, please try the GENERIC kernel before sending in a bug report. 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.

5.3 - Kernel Configuration Options

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 more complete list of options. You can also check the example configuration files that are available for your architecture.

Not all kernel options have been tested for compatibility with all other options. Don't put an option in your kernel unless you actually have a reason to do so! The one kernel configuration which gets the most testing is the GENERIC kernel. This is usually a combination of the options in `/usr/src/sys/arch/`arch` -s`/conf/GENERIC` and `/usr/src/sys/conf/GENERIC`.

- [Alpha Kernel Conf Files](#)
- [i386 Kernel Configuration files](#)
- [macppc Kernel Configuration files](#)
- [sparc Kernel Configuration Files](#)
- [sparc64 Kernel Configuration Files](#)
- [vax Kernel Configuration Files](#)
- [Other Arch's](#)

Look closely at these files and you will notice a line like:

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

This means that it is referencing yet another configuration file. This file stores non arch-dependent options. So when creating your Kernel Config be sure to look through [/sys/conf/GENERIC](#) and see what you want. There **are** options in there that are **necessary**.

All of the options listed below should be placed in your kernel configuration file in the format of:

```
option OPTION
```

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.

OpenBSD has many compatibility options which allow you to use binaries from other OS's. Not all are available on every architecture, so be sure to read the man pages for each option to see if your arch is supported.

- [COMPAT_SVR4\(8\)](#) - Compatibility with SVR4 binaries.
- [COMPAT_BSDOS\(8\)](#) - Compatibility with BSD/OS binaries.
- [COMPAT_LINUX\(8\)](#) - Compatibility with Linux binaries.
- [COMPAT_SUNOS\(8\)](#) - Compatibility with SunOS binaries.
- [COMPAT_ULTRIX\(8\)](#) - Compatibility with Ultrix binaries.
- [COMPAT_FREEBSD\(8\)](#) - Compatibility with FreeBSD binaries.
- [COMPAT_HPUX\(8\)](#) - Compatibility with HP-UX binaries. Only available on some m68k arch's.
- [COMPAT_IBCS2\(8\)](#) - Compatibility to run ibcs2 binaries.
- [COMPAT_OSF1\(8\)](#) - Run Digital Unix binaries. Available only on Alpha platform.
- `COMPAT_43` - Compatibility with 4.3BSD. Use of this is discouraged, but it is needed for some applications.
- `COMPAT_11` - Compatibility with NetBSD 1.1.
- `COMPAT_NOMID` - Compatibility with a.out executables that lack a machine id.

It is always helpful to be able to debug problems with the kernel. But many choose not to put these options in their kernel because these options add considerable size to the kernel. They are however extremely helpful in a case where a bug might be present. This will help the developers discover the source of your problems much quicker. Here is a list of popular debugging options that can be added to your kernel.

- [DDB\(4\)](#) - This compiles in the in-kernel debugger. This isn't available on all platforms. So be sure to read before adding it.
- [KGDB\(7\)](#) - Compiles a remote kernel debugger using gdb's `remote target` feature.
- `makeoptions DEBUG="-g"` - Makes `bsd.gdb` along with `bsd`. This is useful for debugging crash dumps with gdb.
- `DEBUG` - Used to put various debugging options in the kernel, where the source has defined them.
- `KTRACE` - Adds hooks for the system call tracing facility. Which allows users to use [ktrace\(1\)](#).
- `DIAGNOSTIC` - Adds code to the kernel that does internal consistency checks.
- `GPROF` - adds code to the kernel for kernel profiling with [kgmon\(8\)](#).
- `makeoptions PROF="-pg"` - The `-pg` flag causes the kernel to be compiled with support for profiling. Option `GPROF` is required to use this option.

Filesystem Options.

- `FFS` - Berkeley Fast Filesystem **NOTE:** This option is required.
- `EXT2FS` - Second Extended File System. This is needed for those of you who want to read Linux partitions.
- `MFS` - Memory File System that stores files in swappable memory.
- `NFS` - Network File System. This is needed if you will be using NFS.
- `CD9660` - This is iso9660 + rockridge filesystem. This is required to read from CDs.
- `MSDOSFS` - Needed to read MS-DOS FAT filesystems. Also has support for Windows 95 long name + mixed case extensions.
- `FDESC` - Includes code for a file system which can be mounted on `/dev/fd`.
- `KERNFS` - Includes code that permits the mounting of a special file system (normally mounted on `/kern`) in which files representing various kernel variables and parameters may be found.
- `NULLFS` - Code to have a loopback filesystem. [mount_null\(8\)](#) has more information.
- `PROCFS` - Includes code for a special file system (conventionally mounted on `/proc`).
- `PORTAL` - Includes the (experimental) portal filesystem. This permits interesting tricks like opening TCP sockets by opening files in the file system.
- `UMAPFS` - Includes a loopback file system in which user and group ids may be remapped -- this can be useful when mounting alien file systems with different uids and gids than the local system (eg, remote NFS).
- `UNION` - Includes code for the union file system, which permits directories to be mounted on top of each other in such a

way that both file systems remain visible. This code isn't quite stable yet.

- XFS - Add hooks for using a filesystem that is compatible with the AFS filesystem. Currently used by the Arla/AFS code.
- FFS_SOFTUPDATES - Allows for the use of softupdates. To read more on softupdates read the [Softupdates FAQ](#) section.
- NFSSERVER - Allow for the server-side NFS code to be included in the kernel.
- NFSCLIENT - Allow for the client-side NFS code to be included in the kernel.
- FIFO - Support for FIFOs. RECOMMENDED.
- NVNODE=*integer* - Where *integer* is the size of the cache used by the name-to-inode translation routines, (a.k.a. the namei() cache, though called by many other names in the kernel source).
- EXT2FS_SYSTEM_FLAG - This option changes the behavior of the APPEND and IMMUTABLE flags for a file on an EXT2FS filesystem. Read [options\(4\)](#) for more details.
- QUOTA - Support for Filesystem Quota's. To read up on using quotas read [FAQ 10, Quotas](#).

Misc. Options

- PCIVERBOSE - Make the boot process more verbose for PCI peripherals.
- EISAVERBOSE - Make the boot process more verbose for EISA peripherals.
- PCMCIAVERBOSE - Make the boot process more verbose for PCMCIA peripherals.
- APERTURE - Provide in-kernel support for VGA framebuffer mapping by user processes. Needed to run X.
- LKM - Support for Loadable Kernel Modules. Not available on all arch's. Read [lkm\(4\)](#) for more information.
- INSECURE - Hardwires the kernel security level to -1. Read [init\(8\)](#) for more information on kernel security levels.
- RAM_DISK_HOOKS - Allows for machine dependent functions to be called when the ramdisk driver is configured.
- RAM_DISK_IS_ROOT - Forces the ramdisk to be root.
- CCDNBUF=*integer* - Set number of component buffers used by [CCD\(4\)](#). Default is 8. For more on CCD, read the man page, or the [Performance Tuning FAQ section](#).
- KMEMSTATS - This makes malloc(9), the kernel memory allocator, keep statistics on its use. If option DEBUG is used, this option is automatically turned on by config.
- BOOT_CONFIG - Adds support for the -c boot option.

Networking Options

Also check the [Networking FAQ](#) or the [Networking Performance Tuning FAQ](#).

- GATEWAY - Enables IPFORWARDING and (on most ports) increases the size of NMBCLUSTERS.
- NMBCLUSTERS=*integer* - Controls the size mbuf cluster map.
- IPFORWARDING - Enables IP routing behavior. With this option enabled, the machine will forward IP datagrams between its interfaces that are destined for other machines.
- MROUTING - Includes support for IP multicast routers.
- INET - Includes support for the TCP/IP protocol stack. This option is REQUIRED.
- MCLSHIFT=*value* - This option is the base-2 logarithm of the size of mbuf clusters. Read [options\(4\)](#) for more information on this option.
- NS - Include support for the Xerox XNS protocol stack. See [ns\(4\)](#).
- ISO,TPIP - Include support for the ubiquitous OSI protocol stack. See [iso\(4\)](#) for more information.
- EON - Include support for OSI tunneling over IP.
- CCITT,LLC,HDLCL - Include support for the X.25 protocol stack.
- IPX, IPXIP - Include support for Internetwork Packet Exchange protocol commonly in use by Novell NetWare.
- NETATALK - Include kernel support for the AppleTalk family of protocols.
- TCP_COMPAT_42 - Use of this option is extremely discouraged, so it should not be enabled. TCP bug compatibility with 4.2BSD. In 4.2BSD, TCP sequence numbers were 32-bit signed values. Modern implementations of TCP use unsigned values.
- TCP_NEWRENO - Turns on NewReno fast recovery phase, which allows one lost segment to be recovered per round trip time.
- TCP_SACK - Turns on selective acknowledgements.
- TCP_FACK - Turns on forward acknowledgements allowing a more precise estimate of outstanding data during the fast recovery phase by using SACK information. This option can be used together with TCP_SACK.
- PPP_FILTER - This option turns on [pcap\(3\)](#) based filtering for ppp connections.
- PPP_BSDCOMP - PPP BSD compression.

- PPP_DEFLATE - Used in conjunction with PPP_BSDCOMP.
- IPSEC - This option enables IP security protocol support. See [ipsec\(4\)](#) for more details. This now implies option KEY, which gives support for PFKEYv2.
- ENCDEBUG - This option enables debugging information to be conditionally logged in case IPSEC encounters errors.

SCSI Subsystem Options

- SCSITERSE - Tenser SCSI error messages. This omits the table for decoding ASC/ASCQ info, saving about 8 bytes or so.
- SCSIDEBUG - Prints extra debugging info for the SCSI subsystem to the console.

5.4 - Building your own kernel

Full instructions for creating your own custom kernel are in the [afterboot\(8\)](#) man page.

To compile your kernel from the cdrom you need to first have the source code available. The source is available on both the [official CD](#) (disk 3) and on the [FTP sites](#). The following example assumes that CD3 is mounted on /mnt:

```
# cd /usr/src
# tar xvzf /mnt/src.tar.gz
```

Note: If you are downloading from the FTP servers, you will find TWO files, *src.tar.gz* and *srcsys.tar.gz*. The first is the "userland" -- everything but the kernel, the second is the kernel source. Download and extract both of them as above, as for most uses, you will want both. On the CDROM, they are combined into one file.

Now to create your custom kernel it is easiest to start with the GENERIC kernel. This is located at `/usr/src/sys/arch/$ARCH/conf/GENERIC`, where \$ARCH is your architecture. There are other sample configurations available in that directory as well. Here are two examples for compiling your kernel. The first example is compiling your kernel on a read-only source tree. The second on a writable source tree.

```
# cd /somewhere
# cp /usr/src/sys/arch/$ARCH/conf/SOMEFILE .
# vi SOMEFILE (to make the changes you want)
# config -s /usr/src/sys -b . SOMEFILE
```

followed by either:

```
# make depend
- OR -
# make clean
# make
```

You must run 'make depend' when you have made any changes (including updates and patches) to your source tree (in other words, almost always, unless you need to run 'make clean').

If you have made changes to your kernel configuration options, and/or made major changes to your source tree, you should use 'make clean' instead of the above 'make depend'. Note that it is always safe to do a 'make clean', though it may result in longer compile times, as more will be rebuilt.

To compile a kernel inside a writable source tree do the following:

```
# cd sys/arch/$ARCH/conf
# vi SOMEFILE (to make any changes you want)
# config SOMEFILE (read more about it here: config\(8\))
# cd ../compile/SOMEFILE
# make
```

Where \$ARCH is the architecture you are using (e.g. i386). You can also do a **make depend** to make the dependencies for the next time you compile your kernel.

To move your kernel into place.

```
# cp /bsd /bsd.old
# cp /sys/arch/$ARCH/compile/SOMEFILE/bsd /bsd
```

To revert to your old kernel at boot you just need to

```
boot> bsd.old
```

and your old kernel will be loaded instead of /bsd.

Sometimes when you build a new kernel you will be required to install new bootblocks. To do so, read [FAQ 14, Installing Bootblocks](#), which will give you an overview on using OpenBSD's Bootloader.

5.5 - 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 by fixing and recompiling your kernel. 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, at boot time us the -c option.

```
boot> boot wd0a:/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 get your device configured, use quit or exit and continue booting. After doing so you should correct your Kernel configuration and Compile a new kernel. Refer to [Building your own kernel](#) for help.

5.6 - 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.

5.7 - 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 3.2 (GENERIC) #25: Thu Oct  3 19:51:53 MDT 2002
  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
      nmbclust             [number]         Show/change NMBCLUSTERS
      cachepct             [number]         Show/change BUFCACHEPERCENT
      nkmempg              [number]         Show/change NKMEMPGS
      shmseg               [number]         Show/change SHMSEG
      shmmaxpgs           [number]         Show/change SHMMAXPGS
```

```

ukc> list
  0 audio* at sb0|sb*|gus0|pas0|sp0|ess*|wss0|wss*|ym*|eap*|eso*|sv*|neo*|cmpci*
|clcs*|clct*|auich*|autri*|auvia*|fms*|uaudio*|maestro*|esa*|yds*|emu* flags 0x0
  1 midi* at sb0|sb*|opl*|opl*|opl*|opl*|ym*|mpu*|autri* flags 0x0
  2 nsphy* at aue*|xe*|ef*|gx*|stge*|bge*|nge*|sk*|ste*|sis*|sf*|wb*|tx*|tl*|vr*
|ne0|ne1|ne2|ne*|ne*|ne*|dc*|dc*|rl*|fxp*|fxp*|xl*|xl*|ep0|ep0|ep0|ep*|ep*|ep*|e
p*|ep* phy -1 flags 0x0
  3 nsphyter* at aue*|xe*|ef*|gx*|stge*|bge*|nge*|sk*|ste*|sis*|sf*|wb*|tx*|tl*|
vr*|ne0|ne1|ne2|ne*|ne*|ne*|dc*|dc*|rl*|fxp*|fxp*|xl*|xl*|ep0|ep0|ep0|ep*|ep*|ep
*|ep*|ep* phy -1 flags 0x0
  4 qsphy* at aue*|xe*|ef*|gx*|stge*|bge*|nge*|sk*|ste*|sis*|sf*|wb*|tx*|tl*|vr*
|ne0|ne1|ne2|ne*|ne*|ne*|dc*|dc*|rl*|fxp*|fxp*|xl*|xl*|ep0|ep0|ep0|ep*|ep*|ep*|e
p*|ep* phy -1 flags 0x0
  5 inphy* at aue*|xe*|ef*|gx*|stge*|bge*|nge*|sk*|ste*|sis*|sf*|wb*|tx*|tl*|vr*
|ne0|ne1|ne2|ne*|ne*|ne*|dc*|dc*|rl*|fxp*|fxp*|xl*|xl*|ep0|ep0|ep0|ep*|ep*|ep*|e
p*|ep* phy -1 flags 0x0
  6 iophy* at aue*|xe*|ef*|gx*|stge*|bge*|nge*|sk*|ste*|sis*|sf*|wb*|tx*|tl*|vr*
|ne0|ne1|ne2|ne*|ne*|ne*|dc*|dc*|rl*|fxp*|fxp*|xl*|xl*|ep0|ep0|ep0|ep*|ep*|ep*|e
p*|ep* phy -1 flags 0x0
  7 eeppy* at aue*|xe*|ef*|gx*|stge*|bge*|nge*|sk*|ste*|sis*|sf*|wb*|tx*|tl*|vr*
|ne0|ne1|ne2|ne*|ne*|ne*|dc*|dc*|rl*|fxp*|fxp*|xl*|xl*|ep0|ep0|ep0|ep*|ep*|ep*|e
p*|ep* phy -1 flags 0x0
  8 exphy* at aue*|xe*|ef*|gx*|stge*|bge*|nge*|sk*|ste*|sis*|sf*|wb*|tx*|tl*|vr*
|ne0|ne1|ne2|ne*|ne*|ne*|dc*|dc*|rl*|fxp*|fxp*|xl*|xl*|ep0|ep0|ep0|ep*|ep*|ep*|e
p*|ep* phy -1 flags 0x0
[...snip...]
ukc> disable ep
  65 ep0 disabled
  66 ep* disabled
  67 ep* disabled
 149 ep0 disabled
 150 ep0 disabled
 151 ep* disabled
 152 ep* disabled
 202 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 3.2 (GENERIC) #25: Thu Oct  3 19:51:53 MDT 2002
  deraadt@i386.openbsd.org:/usr/src/sys/arch/i386/compile/GENERIC
Processing history...
105 wi* disabled
106 wi* disabled
Enter 'help' for information
ukc> quit

```

[\[FAQ Index\]](#) [\[To Section 4 - Installation Guide\]](#) [\[To Section 6 - Networking\]](#)



www@openbsd.org

\$OpenBSD: faq5.html,v 1.80 2003/04/04 17:49:08 nick Exp \$



[\[FAQ Index\]](#) [\[To Section 5 - Kernel configuration and Disk Setup\]](#) [\[To Section 7 - Keyboard controls\]](#)

6 - Networking

Table of Contents

- [6.0.1 - Before we go any further](#)
 - [6.1 - Initial network setup](#)
 - [6.2 - Packet Filter \(PF\)](#)
 - [6.3 - Network Address Translation](#)
 - [6.4 - Dynamic Host Configuration Protocol](#)
 - [6.5 - Point to Point Protocol](#)
 - [6.6 - Tuning networking parameters](#)
 - [6.7 - Using NFS](#)
 - [6.8 - Domain Name Service - DNS, BIND, and named](#)
 - [6.9 - Setting up a PPTP connection in OpenBSD](#)
 - [6.10 - Setting up a bridge with OpenBSD](#)
-

6.0.1 - 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.

6.1 - Initial Network Setup

6.1.1 - Identifying and Setting Up Your Network Interfaces

To start off, you must first identify your network interface. 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:f7
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).

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 -a
lo0: flags=8049<UP,LOOPBACK,RUNNING,MULTICAST> mtu 33224
```

```

        inet6 fe80::1%lo0 prefixlen 64 scopeid 0x4
        inet6 ::1 prefixlen 128
        inet 127.0.0.1 netmask 0xff000000
lo1: flags=8008<LOOPBACK,MULTICAST> mtu 33224
fxp0: flags=8843<UP,BROADCAST,RUNNING,SIMPLEX,MULTICAST> mtu 1500
        media: Ethernet autoselect (100baseTX full-duplex)
        status: active
        inet 10.0.0.38 netmask 0xfffff00 broadcast 10.0.0.255
        inet6 fe80::202:b3ff:fe2b:10f7%fxp0 prefixlen 64 scopeid 0x1
pflog0: flags=0<> mtu 33224
sl0: flags=c010<POINTOPOINT,LINK2,MULTICAST> mtu 296
sl1: flags=c010<POINTOPOINT,LINK2,MULTICAST> mtu 296
ppp0: flags=8010<POINTOPOINT,MULTICAST> mtu 1500
ppp1: flags=8010<POINTOPOINT,MULTICAST> mtu 1500
tun0: flags=10<POINTOPOINT> mtu 3000
tun1: flags=10<POINTOPOINT> mtu 3000
enc0: flags=0<> mtu 1536
bridge0: flags=0<> mtu 1500
bridge1: flags=0<> mtu 1500
vlan0: flags=0<> mtu 1500
vlan1: flags=0<> mtu 1500
gre0: flags=8010<POINTOPOINT,MULTICAST> mtu 1450
gif0: flags=8010<POINTOPOINT,MULTICAST> mtu 1280
gif1: flags=8010<POINTOPOINT,MULTICAST> mtu 1280
gif2: flags=8010<POINTOPOINT,MULTICAST> mtu 1280
gif3: flags=8010<POINTOPOINT,MULTICAST> mtu 1280

```

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 0xfffff00 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
- [sl](#) - SLIP Network Interface
- [ppp](#) - Point to Point Protocol
- [tun](#) - Tunnel Network Interface
- [enc](#) - Encapsulating Interface
- [bridge](#) - Ethernet Bridge Interface
- [vlan](#) - IEEE 802.1Q Encapsulation Interface
- [gre](#) - GRE/MobileIP Encapsulation Interface
- [gif](#) - Generic IPv4/IPv6 Tunnel Interface

If you don't have your interface configured, the first step is to create the `/etc/hostname.xxx` file, where the name of your interface will take the place of xxx. From the information in the examples above, the name would be `/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.)

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
```

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.)

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 172.21.0.0 255.255.255.0 NONE vlan 2 vlandev fxp1
```

The next step from here is to setup your default gateway. To do this, simply put the IP of your gateway in the file `/etc/mygate`. This will allow for your gateway to be set upon boot. From here you should setup your nameservers, and your `/etc/hosts` file (see the [hosts\(5\)](#) man page). To setup your nameservers, you will create a file called `/etc/resolv.conf`. You can read more about the format of this file in the [resolv.conf\(5\)](#) man page. But for a standard usage, here is an example. In this example your domain servers are 125.2.3.4 and 125.2.3.5. You also belong in the domain "yourdomain.com".

```
$ cat /etc/resolv.conf
search yourdomain.com
nameserver 125.2.3.4
nameserver 125.2.3.5
lookup file bind
```

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\(8\)](#). You can also 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
```

6.1.2 - Setting up your OpenBSD box as a 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 -w net.inet.ip.forwarding=1
net.inet.ip.forwarding: 0 -> 1
```

Now modify the routes on the other hosts on both sides. There are many possible uses of OpenBSD as a router, using software such as [routed\(8\)](#), [gated](#), [mrtld](#), and [zebra](#). OpenBSD has support in the ports collection for zebra, gated and mrtld. OpenBSD supports several T1, HSSI, ATM, FDDI, Ethernet, and serial (PPP/SLIP) interfaces.

6.1.3 - 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/rc(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 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
```

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 0xfffff00 broadcast 192.168.0.255
    inet 192.168.0.3 netmask 0xffffffff broadcast 192.168.0.3
```

6.2 - Packet Filter (PF)

Note: For OpenBSD 3.2, the functions of the `/etc/nat.conf` have been merged into `/etc/pf.conf`.

The new [/etc/pf.conf](#) file has four parts:

- **Options:** Various options to control how PF works.
- **Scrub:** Reprocessing packets to normalize and defragment them.
- **NAT and Redirection Rules:** NAT allows many machines to access the Internet through one IP address. Redirection allows incoming requests to be forwarded to a particular machine behind the NAT.
- **Filter Rules:** Allows the selective filtering or blocking of packets as they pass through any of the interfaces.

None of these sections are required to exist, but those that do must be in the above order.

The Packet Filter subsystem was created to handle two tasks, dealing with packet level forwarding permissions and mapping hosts/subnets to a range of external addresses. The configuration file for this service is [/etc/pf.conf\(5\)](#).

To start these services with your system, you need to edit [/etc/rc.conf\(8\)](#) and set the line to:

```
pf=YES
```

If you are using NAT, you will most likely also need to set the [sysctl\(8\)](#) value `net.inet.ip.forwarding` to 1. You can do this by uncommenting the relevant lines in [/etc/sysctl.conf\(5\)](#) and rebooting your computer.

If you have Packet Filter compiled into your kernel, but you don't have it turned on in your [/etc/rc.conf\(8\)](#) file, you can still activate it with the [pfctl\(8\)](#) command.

```
# pfctl -f /etc/pf.conf
# pfctl -e
```

The first line sets up filtering and NAT, using [/etc/pf.conf](#) and the second line activates PF.

This can also be combined into one command line:

```
# pfctl -f /etc/pf.conf -e
```

If you make changes to [/etc/pf.conf](#) after PF is started, you can reload your rules by reloading the file:

```
# pfctl -f /etc/pf.conf
```

This document will cover some basic [pf.conf\(5\)](#) configurations below. You can also look at the [resulting ruleset](#) that includes all the tweaks explained below in more detail. You can find other Packet Filter information at the [Packet Filter web site](#) and in the [Packet Filter HOWTO](#).

Packet Filter

In order to enable Packet Filter at boot, you will need to modify [/etc/rc.conf](#) so that it reads `pf=YES`. Packet Filter (pf) is controlled by [/etc/pf.conf](#), which is read at boot. For a more detailed explanation, see [pf.conf\(5\)](#). In the examples that follow, `fxp0` will represent the external interface to the internet. It will be different for you, based on the ethernet adapter present in your computer. These rules will assume full-time internet connectivity, such as you would see on a webserver.

Packet Filter rules are processed sequentially from top to bottom; it helps to visualize each packet having to traverse every rule before it reaches its destination.

For example, the default ruleset provided allows all packets to travel in, and all packets to travel out:

```
pass out all
pass in all
```

This is shorthand, for:

```
pass in from any to any
pass out from any to any
```

which can be read as "pass incoming packets from any source to any destination", with an implied "on any interface (which is always implied if a specific interface isn't specified) of any address family, [inet \(v4\)](#) or [inet6 \(v6\)](#)".

Obviously, this isn't much of a filter. More useful filtering will be based on the address family (IPv4 or IPv6), protocol(s) and port(s) used by the services you wish to filter. Any of the protocols listed in [/etc/protocols\(5\)](#) can be specified, by either name or number, but we will concern ourselves with [tcp\(4\)](#), [udp\(4\)](#) and [icmp\(4\)](#).

Now let's say we don't want to allow any incoming IPv4 connections to TCP port 3306 (MySQL) because the database should only be connected to from localhost. Our ruleset would look like this:

```
pass out all
pass in all
block in on fxp0 inet proto tcp from any to any port 3306
```

This says "block in all IPv4 (inet) incoming packets, from any source to any destination bound for tcp port 3306." Note that it is required that you specify a protocol with any port-based filter, and recommended that you specify the address family. For services defined in the [/etc/services\(5\)](#) file, you can also use the service names, such as `www` or `mysql`. A packet destined for tcp port 3306 on interface `fxp0` will pass the first "pass in" rule and then be dropped by the "block in port 3306" rule. If you reversed the order of our incoming rules (remember, order is important):

```
pass out all
block in on fxp0 inet proto tcp from any to any port 3306
```

```
pass in all
```

Packets destined for tcp port 3306 would pass because the last rule in the set allows all packets to pass. It is important to keep this in mind when writing packet filter rules: **The last matching rule wins**.

Of course, there are exceptions to every rule. The *quick* option drops the packet at the first rule that matches. Let's look at our above flawed example, if we add *quick* to the "block in" rule:

```
pass out all
block in quick on fxp0 inet proto tcp from any to any port 3306
pass in all
```

A packet destined for our host on tcp port 3306 will hit the "block in quick" rule and be dropped immediately. All packets destined for other ports or protocols won't find a rule match until they reach our "pass in" rule that allows all packets to pass.

Default Deny

The safest packet filtering policy is a default deny policy. All traffic not explicitly allowed is denied. This policy is far safer than explicitly denying each protected service, allows for smaller rulesets, and can protect from an accidentally misconfigured service that has been left exposed.

Let's now look at another example ruleset and explain things line by line. Here's an example for a webserver with a default deny policy that only allows ssh connections (for administration) and connections to http (port 80) and https (port 443).

```
block in on fxp0 all
pass in on fxp0 inet proto tcp from any to any port 22
pass in on fxp0 inet proto tcp from any to any port 80
pass in on fxp0 inet proto tcp from any to any port 443
pass out on fxp0 all
```

This will allow incoming connections from anywhere to tcp ports 22(ssh), 80(http), and 443(https). It will drop all other connection attempts, and allow all outgoing connections. This is a pretty tight ruleset. But what if you only wanted to allow internal hosts on your 1.1.1.0 address block to connect to ssh, but allow outside connections to http and https?

```
block in on fxp0 all
pass in on fxp0 inet proto tcp from 1.1.1.0/24 to any port 22
pass in on fxp0 inet proto tcp from any to any port 80
pass in on fxp0 inet proto tcp from any to any port 443
pass out on fxp0 all
```

Pretty good, but what if we only want to allow one machine (1.1.1.1) to administer the web server remotely? In that case, we can change this:

```
pass in on fxp0 inet proto tcp from 1.1.1.0/24 to any port 22
```

to this:

```
pass in on fxp0 inet proto tcp from 1.1.1.1/32 to any port 22
```

Sample Rules

Here are some good rules for everyone to use (assuming that fxp0 is the external internet-connected interface). First we will set up a simple address spoofing protection. These addresses shouldn't (normally) be floating around the Internet, and if they are, it is rarely good, so we block them:

```
block in quick on fxp0 inet from { 127.0.0.0/8, 192.168.0.0/16, \
172.16.0.0/12, 10.0.0.0/8 } to any
block out quick on fxp0 inet from any to { 127.0.0.0/8, 192.168.0.0/16, \
172.16.0.0/12, 10.0.0.0/8 }
```

Our ruleset is starting to look pretty good; when we put it together, here is what it looks like:

```
# don't allow anyone to spoof non-routeable addresses
block in quick on fxp0 inet from { 127.0.0.0/8, 192.168.0.0/16, \
172.16.0.0/12, 10.0.0.0/8 } to any
block out quick on fxp0 inet from any to { 127.0.0.0/8, 192.168.0.0/16, \
172.16.0.0/12, 10.0.0.0/8 }
```

```
# only allow our administration machine to connect via ssh
pass in quick on fxp0 inet proto tcp from 1.1.1.1/32 to any port 22

# allow others to use http and https
pass in quick on fxp0 inet proto tcp from any to any port 80
pass in quick on fxp0 inet proto tcp from any to any port 443

# finally lock the rest down with a default deny
block in quick on fxp0 from any to any

# and let out-going traffic out
pass out on fxp0 from any to any
```

Packet Logging

Now that's pretty good, but it could be better. What if we want to log any connection attempts to port 22(ssh) that get blocked by our firewall? Easy, Packet Filter can handle this with the *log* keyword:

```
pass in quick on fxp0 inet proto tcp from 1.1.1.1/32 to any port 22
block in log quick on fxp0 inet proto tcp from any to any port 22
```

This rule will allow our remote administration machine to connect to port 22, but deny and log all other attempts to connect to port 22.

Logged packets are sent to the pflog0 interface, which is monitored by [pflogd\(8\)](#), which typically dumps the packets to */var/log/pflog* in [tcpdump\(8\)](#) binary format. pflogd(8) is started by default by [/etc/rc\(8\)](#) if pf is enabled in [/etc/rc.conf\(8\)](#). You can read these log files using the following command:

```
# tcpdump -n -e -ttt -r /var/log/pflog
```

One should be aware that using tcpdump to watch the pflog file does NOT give a real-time display. If you wish a real-time display, you can use this command:

```
# tcpdump -i pflog0
```

One could also use tcpdump to narrow down the focus to facilitate debugging:

```
# tcpdump -e -i pflog0 port 80
```

Doing this does NOT impact the data that is put to the file */var/log/pflog*.

When examining one's logs, especial care should be taken with tcpdump's verbose protocol decoding (activated via the *-v* command line option). Tcpdump's protocol decoders do not have a perfect security history. At least in theory, a delayed attack could be possible via the partial packet payloads recorded by the logging device.

Additional care should be taken about access to the logs. Pflogd will capture 96 bytes of the packet and log it. Access to the logs could provide partial access to sensitive packet payloads (like [telnet\(1\)](#) or [ftp\(1\)](#) logins).

Packet Logging through syslog

In many situations it is desirable to have the firewall logs available in ASCII format and/or to send them to a remote logging server. All this can be accomplished with 2 small shell scripts and with minor changes of the OpenBSD configuration files.

[Syslogd\(8\)](#) is the standard daemon for logging, it logs in ASCII and is also able to log to a remote logging server.

First we have to create a user *pflogger* with a *.nologin*. shell. The easiest way to create this user is with [adduser\(8\)](#).

After creating the user *pflogger* create the following two scripts:

/etc/pflogrotate

```
FILE=/home/pflogger/pflog5min.$(date +%Y%m%d%H%M")
kill -ALRM $(cat /var/run/pflogd.pid)
if [ $(ls -l /var/log/pflog | cut -d " " -f 8) -gt 24 ]; then
    mv /var/log/pflog $FILE
    chown pflogger $FILE
    kill -HUP $(cat /var/run/pflogd.pid)
fi
```

```
/home/pflogger/pfl2sysl
```

```
#!/bin/sh
# feed rotated pflog file(s) to syslog
for logfile in /home/pflogger/pflog5min* ; do
    tcpdump -n -e -ttt -r $logfile | logger -t pf -p local0.info
    rm $logfile
done
```

Edit the cron job for user *root*

```
# crontab -u root -e
```

and add the following two lines:

```
# rotate pf log file every 5 minutes
0-59/5 * * * * /bin/sh /etc/pflogrotate
```

Create a cron job for user *pflogger*

```
# crontab -u pflogger -e
```

and add the following two lines:

```
# feed rotated pflog file(s) to syslog
0-59/5 * * * * /bin/sh /home/pflogger/pfl2sysl
```

Add the following line to */etc/syslog.conf*:

```
local0.info    /var/log/pflog.txt
```

If you want to log to a remote log server also add the line:

```
local0.info    @syslogger
```

and make sure host *syslogger* has been defined in the [/etc/hosts\(5\)](#) file.

All logged packets are sent to */var/log/pflog.txt*. If the second line is added too they are sent to the remote logging host *syslogger* as well.

/etc/pflogrotate now processes and then deletes */var/log/pflog* so rotation of *pflog* by [newsyslogd\(8\)](#) is no longer necessary and it should be disabled. However */var/log/pflog.txt* replaces */var/log/pflog* and rotation of it should be activated. Change */etc/newsyslog.conf* as follows:

```
#/var/log/pflog      600    3      250    *      ZB      /var/run/pflogd.pid
/var/log/pflog.txt   600    7      *      24
```

Pf will now log in ASCII to */var/log/pflog.txt*. If so configured in */etc/syslog.conf* it will also log to a remote server. The logging is not immediate but it can take up to about 5-6 minutes (the cron job interval) before the logged packets appear in the file.

Multiple Protocols

What if we need to allow connections to a service running over multiple protocols, such as bind, which uses TCP and UDP? Packet filter lets you lump options together into sets (more on this later):

```
# Pass DNS traffic for BIND
pass in quick on fxp0 inet proto { tcp, udp } from any to any port 53
```

Notice the spaces on both sides of the '{ }' characters. This is neater than the alternative you might otherwise have to use:

```
pass in quick on fxp0 inet proto tcp from any to any port 53
pass in quick on fxp0 inet proto udp from any to any port 53
```


Packet Normalization

Packet Normalization means reassembling fragmented packets and clearing IP options. Some OSs and applications have trouble with abnormal or fragmented packets, and it's in general good to have normalized packets for the filter rules to look at and for the destination hosts to look at. Thus, it is nearly always beneficial to normalize the packets before they are passed on to their ultimate destination. This is done with the **scrub** directive, used as here:

```
scrub in all
```

This does put a very minor additional load on the system, and requires a bit of memory to cache the packet fragments. The advantages of packet normalization almost always outweigh this cost.

IP Options

By default, PF blocks packets with IP options set. This can make the job more difficult for "OS fingerprinting" utilities like nmap. If you have an application that requires the passing of these packets, such as multicast or IGMP, you can use the **allow-opts** directive:

```
pass in quick on fxp0 all allow-opts
```

TCP Flags, established connections and keeping state

Packet Filter can also filter packets based on TCP flags and maintain established connections and connection state. It is recommended that all users who wish to filter packets based on TCP flags understand what role each flag plays. For instance, if you wanted to deny all packets with the FIN, URG, and PSH flags set (like for instance an nmap OS fingerprinting attempt) you could use a rule like this:

```
block in quick on fxp0 inet proto tcp from any to any flags FUP/FUP
```

(Thanks to [Kyle Hargraves](#) for that tip)

Packet Filter's next cool trick is its ability to maintain state. Maintaining state has been described as "not speaking until spoken to"; in other words, once a connection is established, packets no longer have to traverse rulesets. This is a very powerful feature allowing much simpler and more secure rule writing.

For example, let's see how we can apply state to our previous example ruleset (confused yet?). To review, we are allowing management access from our Class C to port 22(ssh) and allowing all incoming web traffic on ports 80(http) and 443(https). We are blocking all other traffic. But, what if I want to [ssh\(1\)](#) out of the webserver?

What if I need to use [lynx\(1\)](#) to look up something in the FAQ? Well, I can't because I have blocked all incoming connections other than those on the specified ports.

While this is the safest route, it can be quite inconvenient. By adding the *keep state* keywords to our "pass out" rule, we can automatically allow incoming packets in response to connections we initiate, such as when web browsing. Remember, we do need to specify what protocol we are keeping state for.

```
block in on fxp0 inet proto tcp all
pass in on fxp0 inet proto tcp from 1.1.1.0/24 to any port 22 keep state
pass in on fxp0 inet proto tcp from any to any port 80 keep state
pass in on fxp0 inet proto tcp from any to any port 443 keep state
pass out on fxp0 inet proto tcp all keep state
```

This little change will dramatically increase the flexibility and security of our ruleset: for instance, in the above ruleset, we are allowing all tcp traffic into ports 80 & 443. We can tighten this up even more. In order for a tcp connection to be established, we only need to allow the initial handshake to occur; once that occurs, we can block traffic to that port and allow our "keep state" rule to manage the connection. To allow the initial handshake to complete, we need only allow packets with the SYN flag set and ACK flag not set. By passing only packets with SYN set, we can prevent many forms of portscanning such as FIN scanning. flags S/SA means: out of flags S (SYN) and A (ACK), only SYN may be set. Other flags aren't investigated. The rules now look like this:

```
block in on fxp0 inet proto tcp all
pass in on fxp0 inet proto tcp from 1.1.1.0/24 to any port 22 \
    flags S/SA keep state
pass in on fxp0 inet proto tcp from any to any port 80 \
    flags S/SA keep state
pass in on fxp0 inet proto tcp from any to any port 443 \
    flags S/SA keep state
block out on fxp0 inet proto tcp all
pass out on fxp0 inet proto tcp all flags S/SA keep state
```

Let's start to tie things together by putting all of the rules we have so far into a ruleset. This ruleset will support IPv4, have a default deny policy, allow management connections from an internal network only (via ssh) and allow incoming traffic on ports 80(http) and 443(https). It will also protect against spoofed non-routeable ip addresses, and drop all packets that are too fragmented to inspect. A pretty comprehensive setup for a public webserver. Here's what */etc/pf.conf* could look like:

```
# Clean up fragmented and abnormal packets
```

```

scrub in all

# don't allow anyone to spoof non-routeable addresses
block in quick on fxp0 inet from { 127.0.0.0/8, 192.168.0.0/16, \
172.16.0.0/12, 10.0.0.0/8 } to any
block out quick on fxp0 inet from any to { 127.0.0.0/8, 192.168.0.0/16, \
172.16.0.0/12, 10.0.0.0/8 }

# by default, block all incoming packets, except those explicitly
# allowed by further rules
block in on fxp0 all

# allow others to use http and https
pass in on fxp0 inet proto tcp from any to any port 80 \
    flags S/SA keep state
pass in on fxp0 inet proto tcp from any to any port 443 \
    flags S/SA keep state

# and let out-going traffic out and maintain state on established connections
# pass out all protocols, including TCP, UDP and ICMP, and create state,
# so that external DNS servers can reply to our own DNS requests (UDP).
block out on fxp0 all
pass out on fxp0 inet proto tcp all flags S/SA keep state
pass out on fxp0 inet proto udp all keep state
pass out on fxp0 inet proto icmp all keep state

```

While this may look good, there are some things Packet Filter will let you do to make your *pf.conf* file look neater and easier to maintain.

Sets

Sets are useful "shortcuts" for writing simple and clear rules in PF. For example, what if we need to allow connections to a service running over multiple protocols, such as BIND, which uses TCP and UDP?

```

pass in quick on fxp0 inet proto { tcp, udp } from any to any port 53

```

Note the spaces on both sides of the '{ }' characters.

Groups of related IPs can be clustered together into sets, which can be used anywhere a single IP could be used. For example, looking at our anti-spoofing rules above:

```

# don't allow anyone to spoof non-routeable addresses
block in quick on fxp0 inet from { 127.0.0.0/8, 192.168.0.0/16, \
172.16.0.0/12, 10.0.0.0/8 } to any
block out quick on fxp0 inet from any to { 127.0.0.0/8, 192.168.0.0/16, \
172.16.0.0/12, 10.0.0.0/8 }

```

Variable Expansion

One problem with the above sample *pf.conf* file is that should you need to change your NIC, or change an IP address, you would need to change a number of lines. This can be lessened by using variable expansion:

```

NoRouteIPs="{ 127.0.0.0/8, 192.168.0.0/16, 172.16.0.0/12, 10.0.0.0/8 }"
ExtIF="fxp0"
block in quick on $ExtIF from $NoRouteIPs to any
block out quick on $ExtIF from any to $NoRouteIPs

```

Putting it all together

Now, let's put it all together, and look at the elegance of the file:

```

# Define useful variables
ExtIF="fxp0"           # External Interface
IntNet="1.1.1.0/24"    # Our internal network
NoRouteIPs="{ 127.0.0.0/8, 192.168.0.0/16, 172.16.0.0/12, 10.0.0.0/8 }"
Services="{ www, https }"

# Clean up fragmented and abnormal packets
scrub in all

# don't allow anyone to spoof non-routeable addresses

```

```

block in quick on $ExtIF from $NoRouteIPs to any
block out quick on $ExtIF from any to $NoRouteIPs

# by default, block all incoming packets, except those explicitly
# allowed by further rules
block in on $ExtIF all

# allow others to use http and https
pass in on $ExtIF inet proto tcp from any to any port $Services \
    flags S/SA keep state

# and let out-going traffic out and maintain state on established connections
# pass out all protocols, including TCP, UDP and ICMP, and create state,
# so that external DNS servers can reply to our own DNS requests (UDP).
block out on $ExtIF all
pass out on $ExtIF inet proto tcp all flags S/SA keep state
pass out on $ExtIF inet proto udp all keep state
pass out on $ExtIF inet proto icmp all keep state

```

If you experience problems, you may want to enable logging on individual rules to effectively troubleshoot. ie:

```
pass in log quick on fxp0 proto tcp from 1.1.1.0/24 to any port 22
```

[pflogd\(8\)](#) will write the ip log entries to `/var/log/pflog`. Remember that `/var/log/pflog` is a binary file, intended to be read by [tcpdump\(8\)](#), NOT directly by humans.

When you modify the configuration file to log packets, don't forget to **pfctl -R /etc/pf.conf** for the changes to take effect!

6.3 - NAT

Note: For OpenBSD 3.2 and later, the NAT functions are incorporated in the `/etc/pf.conf` file, not the separate `/etc/nat.conf` file used in OpenBSD 3.0 and 3.1

6.3.1 NAT Introduction

Based on [RFC 1631](#), NAT provides an easy way to map internal networks to a single routeable ("real") internet address. This is very useful if you don't have officially assigned addresses for every host on your internal network. When you set up private/internal networks, you can take advantage of reserved address blocks (assigned in [RFC 1918](#)), such as:

```

10.0.0.0/8 (10.0.0.0 - 10.255.255.255)
172.16.0.0/12 (172.16.0.0 - 172.31.255.255)
192.168.0.0/16 (192.168.0.0 - 192.168.255.255)

```

The user is assumed to have already set up and configured an OpenBSD machine with two network cards (one connected to the Internet and the other to the local network).

Configuration

For the purpose of example, we will use the system described below. Your setup will almost certainly vary from this, so be very careful in typing anything you see here literally into your system and expecting it to work as you desire.

NICs:

```

Intel EtherExpress Pro/100 fxp0
Connected to the EXTERNAL LAN (or WAN)
IP Address: 24.5.0.5
Netmask: 255.255.255.0

```

```

Compaq Netelligent 10/100Mb tl0
Connected to the INTERNAL LAN
IP Address: 192.168.1.1
Netmask: 255.255.255.0

```

External, Internet-routeable IP (provided by ISP, in this example, a cable modem provider)

```

IP Address: 24.5.0.5
Netmask: 255.255.255.0

```

Gateway: 24.5.0.1

Local Area Network

In this example environment, machines on the internal network use the IP addressing scheme 192.168.1.xxx (where xxx is a unique number). There could be a variety of different operating systems on the internal network, such as Windows 98, Windows NT, OpenBSD and Linux, but the client OS is not an issue for NAT. For the examples, the client we will look at on the internal network will be assumed to have an IP address of 192.168.1.40.

Diagram of Configuration

```

+-----+           +-----+           +-----+
| Hub |----- t10 | NAT | fxp0 ----| Internet |
+-----+           +-----+           +-----+
| |
| +- Client A
+---- More clients

```

```

+-----+
| LEGEND |
+-----+
| NIC fxp0 - 24.5.0.5 |
| NIC t10 - 192.168.1.1 |
| Client A - 192.168.1.40 |
+-----+

```

6.3.2 Network Address Translation

Introduction to NAT

Each node on the Internet requires a unique IP address. At least with IPv4, there is a very finite number of distinct IP addresses available, and as a result, they are not free. Most "low-cost" ISPs will limit a site to anywhere from 1 to 30 addresses, and while larger budget organizations may be able to afford a larger block, in most cases, there are relatively few benefits and considerable risks to having each computer individually addressable on the Internet.

Network Address Translation, or NAT, (also known as "IP Masquerading" if you are coming from a Linux background) allows multiple computers to be located "behind" one (or a small number of) IP address. Each "internal" computer has a locally assigned, unregistered IP address (per [RFC 1918](http://www.ietf.org/rfc/rfc1918.txt)), and all utilize the same external IP address, simultaneously.

The way NAT works is rather simple. When a client on the LAN wants to connect to a machine on the Internet, it sends out a TCP packet with a request to connect. Inside the TCP packet header is the client's IP address (e.g. 192.168.1.40) and the requested host's IP address (e.g. 123.45.67.89). The machine running NAT intercepts this TCP packet and changes the client's IP address from 192.168.1.40 to the IP address of the Internet-connected machine (e.g. 24.5.0.5). This effectively tricks the host machine into thinking the actual connection is from the NAT machine, not the actual client's machine. The host then sends back responses to the NAT machine like it was the one connecting. When the NAT machine receives the responses it quickly translates the destination IP address back from itself to the client's machine and sends the packet to the client. The client normally does not have any idea what happened and the apparent Internet connectivity is transparent to the user and user's applications.

The example below shows NAT a little more clearly:

```

Client ----- t10 [ NAT ] fxp0 ----- Internet Host
192.168.1.40 --- 192.168.1.1 [ NAT ] 24.5.0.5 --- 123.45.67.89

OUTGOING TCP Packet                               OUTGOING TCP Packet
From: 192.168.1.40 >>=== NAT ===>>             From: 24.5.0.5
To: 123.45.67.89                                  To: 123.45.67.89

INCOMING TCP Packet                               INCOMING TCP Packet
From: 123.45.67.89                                From: 123.45.67.89
To: 192.168.1.40 <<=== NAT ===<<              To: 24.5.0.5

```

Why use NAT?

When presented with a cable modem in my new apartment I was also presented with another minor problem. How to get Internet access for my roommates, when the cable modem resides in my room? There were a few options I could implement, ranging from obtaining extra IP addresses, to setting up a proxy server, to setting up NAT. (Don't let the home cable modem example fool you. NAT is powerful enough to masquerade a large network with hundreds or even thousands of computers!)

There are many reasons why I wanted to set up NAT. The number one reason is to save money. There are two roommates in my house; splitting the cost is certainly attractive. Further, as each roommate has their own PC and I have three, we had five computers to connect, but my ISP only allows for three IP addresses per household. This means that there weren't enough IP addresses to allow every machine simultaneous direct Internet access.

By using NAT each machine will have a unique internal IP address but share the one IP address given to me by my ISP. The cost goes down.

Setup

In order to enable NAT on your OpenBSD machine you will need to turn on PF. This is easily accomplished by editing the files listed below (make the changes to the file so it looks like the options below):

/etc/rc.conf (this file used to start services at boot time)

```
pf=YES
```

/etc/sysctl.conf

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

After these changes are made, the machine is now ready for the configuration of NAT.

Configuration

The first step is to configure the PF configuration file ([/etc/pf.conf](#)). For the purposes of this document we will allow traffic to pass through this firewall without any interference. The file should look like this:

```
pass in all
pass out all
```

Again, for more information you can read [FAQ 6, Packet Filter](#)

The NAT portion of the PF configuration file ([/etc/pf.conf](#)) has a very simple syntax. For the configuration set forth above, the file should contain the following entry:

```
nat on fxp0 from 192.168.1.0/24 to any -> 24.5.0.5
```

Here is an explanation for the above lines.

"nat"

This indicates the command you are giving is a NAT rule.

"fxp0"

This is the network interface that is connected to the Internet.

"192.168.1.0/24"

the IP address and netmask (the netmask is in CIDR format). Combined they state "any IP address of value 192.168.1.1 through 192.168.1.254" should be mapped.

"24.5.0.5"

This is the external IP address the internal IP addresses will be mapped to.

Running

Once the configuration is complete, there are two ways to enable NAT. The first (and best way if possible) is to reboot your OpenBSD machine. This is accomplished with the command "*reboot*".

If you would like to configure NAT from the command line, use the following commands:

```
# pfctl -f /etc/pf.conf
# pfctl -e
```

The first line is to load a set of NAT rules into PF (and flushing any old rules); the second line enables PF. Again, though, rebooting is the best way to ensure that everything will startup from a reboot as you expect.

Note: in order to reload the NAT settings (in case you edit the file but don't want to reboot) just execute the first command over again. The settings will be flushed and reloaded.

6.3.3 Nat Knowledge Base

Checking NAT Status

To find out how NAT is doing or make sure the settings have taken effect, you use the "-ss" option. This option will list all the current sessions that NAT is running:

```
# pfctl -ss
TCP 192.168.1.40:2132 -> 24.5.0.5:53136 -> 65.42.33.245:22      TIME_WAIT:TIME_WAIT
TCP 192.168.1.40:2492 -> 24.5.0.5:55011 -> 65.42.33.245:22      ESTABLISHED:ESTABLISHED
UDP 192.168.1.40:2491 -> 24.5.0.5:60527 -> 24.2.68.33:53        2:1
```

Explanations (first line, others are similar):

"192.168.1.40:2132"

This tells you the IP address of the machine on the LAN that is using NAT (192.168.1.40). The port number used to make the connection (2132) is displayed afterwards.

"24.5.0.5:53136"

This denotes that the connection is going to the Internet via IP address 24.5.0.5 and using port 53136.

"65.42.33.245:22"

The IP address and the port being connected to.

"TIME_WAIT:TIME_WAIT"

This indicates what state PF believes the TCP connection to be in.

Issues with FTP and NAT

There are a few limitations of NAT; the most commonly encountered is with FTP. You can use FTP in two ways: passive and active. Of these, passive FTP is generally considered more secure.

With active FTP, when a user connects to a remote FTP server and requests information or a file, the FTP client sends the server a random port number that the FTP server will make a connection to on the client and transfer the info. This is a problem for users attempting to gain access to FTP servers from within the LAN. When the FTP server sends its information it sends it to the external NIC at a random port. The NAT machine will receive this, but because it has no mappings for the unknown packet and doesn't have any mappings for that port, it will drop the packet and won't deliver it.

With passive mode FTP (the default with OpenBSD [ftp\(1\)](#) client), the client requests that the server picks up a random port that it will listen on for the data connection. The server informs the client of the port it has chosen, and the client connects to this port to transfer the data. Unfortunately, this is not always possible or desirable. [ftp\(1\)](#) uses this mode by default; to force active mode FTP, use the -A flag to ftp, or set the passive mode to off by issuing the command

```
passive off
```

at the ftp> prompt.

Packet Filter provides another solution for this situation, redirecting FTP traffic through an FTP proxy server, a process which acts to "guide" your FTP traffic through the filters. The FTP proxy used by OpenBSD and PF is [ftp-proxy\(8\)](#). To activate it, put something like this in your */etc/pf.conf* file:

```
rdr on tl0 proto tcp from any to any port 21 -> 127.0.0.1 port 8021
```

Short explanation of this line is, "TCP port 21 traffic on the internal interface is redirected to the proxy server running on this machine which is listening at port 8021".

Hopefully, it is apparent the proxy server has to be started and running on the OpenBSD box; this is done by inserting the following line in */etc/inetd.conf*:

```
127.0.0.1:8021 stream tcp nowait root /usr/libexec/ftp-proxy ftp-proxy
```

and either rebooting the system or sending a 'HUP' signal to [inetd\(8\)](#). One way to send the 'HUP' signal is with the command:

```
kill -HUP `cat /var/run/inetd.pid`
```

You will note that ftp-proxy is listening on port 8021, the same port the above rdr statement was sending FTP traffic to. The choice of port 8021 is arbitrary, though 8021 is a good choice, as it is not defined for any other application.

Redirecting Traffic

For some applications, you may need to redirect incoming or outgoing traffic for a certain protocol and/or port to a particular machine behind the filtering system. An example of this would be a computer residing inside the local network running a web server which was to be accessed by the outside world (or of course, the already discussed ftp-proxy(8)). Incoming connections to your valid Internet IP will find that unless your NAT box is also running a web server, no connection can be made. For this purpose we use the NAT 'rdr' directive in the rules file to instruct where to redirect a particular connection to.

For our example, lets say a web server resides on the LAN with IP address of 192.168.1.80. The NAT rules file needs a new directive to handle this. Add a line similar to the following one to your */etc/pf.conf*:

```
rdr on fxp0 proto tcp from any to any port 80 -> 192.168.1.80 port 80
```

The explanation for each part of this line:

"rdr"

This is the command you are giving NAT. It is telling NAT that this entry is an entry to redirect a connection.

"on fxp0"

This is the network interface that is connected to the Internet.

"from any to any"

This indicates which IP addresses to redirect (from any coming in on fxp0, as indicated above, to any destination IP)

"port 80"

This is the incoming port (80) that should be redirected. The number "80" didn't have to be used. You can use "port www" also to specify a redirection of port 80. If you would like to use a name instead of a number, the service name and corresponding port must exist in the file */etc/services*.

"192.168.1.80 port 80"

The IP address of the LAN machine which the packets are redirected to. Note that the destination port does NOT need to match the incoming port. For example, the following is valid, and even potentially useful:

```
rdr on fxp0 proto tcp from any to any port 8080 -> 192.168.1.35 port 80
```

This line would redirect incoming traffic on port 8080 to a webserver running on a machine in the internal network, at the "standard" port 80.

When the addition is complete reload the NAT rules, and the redirection will start immediately.

Negation

Sometimes, you need to make exceptions to a NAT or redirection rule. Here's an example. AOL Instant Messenger is noted for sneaking out firewalls through any available port. You may find that the ftp-proxy is interfering with AIM when it chooses to go out to remote port 21. In the event you consider this bad (many people spend considerable time trying to block AIM!), you might wish to exclude the IP addresses used by the AIM servers from the traffic redirected by our above ftp-proxy line. You can do this with the following line:

```
rdr on tl0 proto tcp from any to ! 64.12.163.199 port 21 -> 127.0.0.1 port 8021
```

Interpretation: Redirect traffic coming in on tl0 going to port 21 but NOT to 64.12.163.199 (the AIM server users were having trouble with) to localhost port 8021 (where hopefully ftp-proxy is waiting). Now, be advised that there are many AIM servers; if this application interests you, you will probably have to play with these IP addresses (64.12.0.0/16 might be more productive, though probably also interferes with some non-AOL sites).

NAT versus Proxy

The difference between NAT and an application-based proxy is that the proxy software acts as a middle-man between the Internet and the machines connected on the LAN. This is fine, however each application you want to run on your machine and connect to the Internet through the proxy server MUST be proxy-aware (be able to use a proxy server). Not all applications are able to do this (especially games). Furthermore, there simply are not proxy server applications for all of the Internet services out there. NAT transparently maps your internal network so that it may connect to the Internet. The only security advantage to using a proxy software over NAT is that the proxy software may have been made security aware, and can filter based on content, to keep your Windows machine from getting a macro virus, it can protect against buffer overflows to your client software, and more. To maintain these filters is often a high-maintenance job.

Redirection and reflection

Often, redirection rules are used to forward incoming connections from the Internet to a local server with a private address in the LAN, as in:

```
rdr on $ext_if proto tcp from any to $ext_if port 80 -> $server port 80
```

But when the redirection rule is tested from a client on the LAN, it's found to not work. The reason is that redirection rules apply only to packets that pass through the specified interface (`$ext_if`, the external interface, in the example). Connecting to the external address of the firewall from a host on the LAN, however, does not mean the packets will actually pass through its external interface. The TCP/IP stack on the firewall compares the destination address of incoming packets with its own addresses and aliases, and detects connections to itself as soon as they have passed the internal interface. Such packets do not physically pass through the external interface, and the stack does not simulate such a passage in any way. `pf` never sees these packets on the external interface, and the redirection rule, specifying the external interface, does not apply.

Adding a second redirection rule for the internal interface does not have the desired effect. When the local client connects to the external address of the firewall, the initial packet of the TCP handshake reaches the firewall through the internal interface. The redirection rule does apply and the destination address gets replaced with that of the internal server. The packet gets forwarded back through the internal interface and reaches the internal server. But the source address has not been translated, and still contains the local client's address, so the server sends its replies directly to the client. The firewall never sees the reply and has no chance to properly reverse the translation. The client receives a reply from a source it never expected, and drops it, the TCP handshake fails, and no connection can be established.

Still, it's often desirable for clients on the LAN to connect to the same internal server as external clients, transparently. There are several solutions for this problem:

Split horizon DNS

It's possible to configure DNS servers to answer queries from local hosts differently than external queries, so that local clients will receive the internal server's address during name resolution. They will then connect directly to the local server, and the firewall isn't involved at all. This reduces local traffic, since packets don't have to be sent through the firewall.

Moving the server into a separate local network

Adding an additional network interface to the firewall and moving the local server from the client's network into a dedicated network (DMZ) allows to redirect connections from local clients in the same way as the redirection of external connections. Use of separate networks has several advantages, including improving security by isolating the server from the remaining local hosts. Should the server (which in our case is reachable from the Internet) ever become compromised, it can't access other local hosts directly, as all connections have to pass through the firewall.

TCP proxying

A generic TCP proxy can be setup on the firewall, either listening on the port to be forwarded or getting connections on the internal interface redirected to the port it's listening on. When a local client connects to the firewall, the proxy accepts the connection, establishes a second connection to the internal server, and forwards data between those two connections.

Simple proxies can be created using [inetd\(8\)](#) and [nc\(1\)](#). The following `/etc/inetd.conf` entry creates a listening socket bound to the loopback address and port 5000. Connections are forwarded to port 80 on server 192.168.1.10.

```
127.0.0.1:5000 stream tcp wait nobody /usr/bin/nc nc -w 20 192.168.1.10 80
```

The following redirection rule forwards port 80 on the internal interface to the proxy:

```
rdr on $int_if proto tcp from $int_net to $ext_if port 80 -> 127.0.0.1 port 5000
```

RDR and NAT combination

With an additional NAT rule on the internal interface, the lacking source address translation described above can be achieved.

```
rdr on $int_if proto tcp from $int_net to $ext_if port 80 -> $server
no nat on $int_if proto tcp from $int_if to $int_net
nat on $int_if proto tcp from $int_net to $server port 80 -> $int_if
```

This will cause the initial packet from the client to be translated again when it's forwarded back through the internal interface, replacing the client's source address with the firewall's internal address. The internal server will reply back to the firewall, which can reverse both NAT and RDR translations when forwarding to the local client. This construct is rather complex, as it creates two separate states for each reflected connection. Care must be taken to prevent the NAT rule from applying to other traffic, for instance connections originating from external hosts (through other redirections) or the firewall itself. Note that the `rdr` rule above will cause the TCP/IP stack to see packets arriving on the internal interface with a destination address inside the internal network. To prevent the stack from issuing ICMP redirect messages (telling the client that its destination is reachable directly, breaking the reflection), disable redirects on the gateway, using

```
# sysctl -w net.inet.ip.redirect=0
```


In general, the previously mentioned solutions should be used instead.

6.3.4 Links and Cross-References

OpenBSD files:

- `/etc/pf.conf` - PF/NAT rules file
- `/etc/rc.conf` - need to edit to start up NAT and PF at boot time
- `/etc/sysctl.conf` - need to edit to enable IP forwarding

NAT Internet Links:

- [pf.conf man page](#)
- [pfctl man page](#)
- <http://www.geektools.com/rfc/rfc1631.txt>

6.4 - DHCP

6.4.1 DHCP Client

To use the DHCP client [dhclient\(8\)](#) included with OpenBSD, edit `/etc/hostname.x10` (this is assuming your main ethernet interface is `x10`. Yours might be `ep0` or `fxp0` or something else!) All you need to put in this hostname file is 'dhcp'

```
# echo dhcp >/etc/hostname.x10
```

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 that you want to receive dhcp on.

No matter how you start the `dhclient`, 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 `hostname`, or other settings too.

6.4.2 DHCP Server

If you want to use OpenBSD as a DHCP server [dhcpcd\(8\)](#), edit `/etc/rc.conf`. Set it up so that `dhcpcd_flags="-q"` instead of `dhcpcd_flags=NO`. Put the interfaces that you want `dhcpcd` to **listen** on in `/etc/dhcpcd.interfaces`.

```
# echo x11 x12 x13 >/etc/dhcpcd.interfaces
```

Then, edit `/etc/dhcpcd.conf`. The options are pretty self-explanatory.

```
option domain-name "xyz.mil";
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 `xyz.mil` (so, if the user types in 'telnet joe' then it will send them to `joe.xyz.mil`). 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 `dhcpcd(8)` from the command line, after editing `/etc/dhpcd.conf`, try:

```
# dhcpcd -q fxp0
```

Where `fxp0` is an interface that you want to start serving DHCP on. The `-q` flag makes `dhcpcd(8)` quiet; otherwise it is very noisy.

If you are serving DHCP to a Windows box, you may want `dhcpcd(8)` to give the client a 'WINS' server address. To make this happen, just the following line to your `/etc/dhpcd.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

Point-to-Protocol is generally what is used to create a connection to your ISP via your modem. OpenBSD has 2 ways of doing this.

- [pppd\(8\)](#) - Which is the kernel ppp daemon.
- [ppp\(8\)](#) - Which is the userland ppp daemon.

The first one we will cover will be the userland PPP daemon. 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 dialup number
- Your nameserver
- Your username and password
- Your gateway

Some of these you can do without, but would be helpful in setting up your 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 though that directory.

Also, make sure, that if you're not using a GENERIC kernel, that you have this line in your configuration file:

```
pseudo-device tun 2
```

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` in which you can simply edit to create your own `ppp.conf` file. Here I will start with the simplest setup and probably most used setup. Here is a quick `ppp.conf` file that will simply connect to your ISP and set your default routes and nameserver. With this file all the information you need is your ISP's phone number and your username and password.

```
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 40 CONNECT"
```

The section under the `default:` tag will get executed each time. Here we setup all our critical information. Here 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 dialup connection and with "set dial" we set our dialup 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 setup our 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 setup essential info for that specific ISP. The first option "set phone" sets your ISP's dialup 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. 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*.

Using PPP(8)

Now that we have our *ppp.conf* file setup 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.

By using `/usr/sbin/ppp` with no options will put you into interactive mode. From here you can interact directly with the modem, it is great for debugging problems in your *ppp.conf* file.

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](#)

Extended information can be found at [FreeBSD Handbook entry on User PPP](#).

6.6 - Tuning networking parameters

6.6.1 - How can I tweak the kernel so that there are a higher number of retries and longer timeouts for TCP sessions?

You would normally use this to allow for routing or connection problems. Of course, for it to be most effective, both sides of the connection need to use similar values.

To tweak this, use `sysctl` and increase the values of:

```
net.inet.tcp.keepinittime
net.inet.tcp.keeppidle
net.inet.tcp.keepintvl
```

Using `sysctl -a`, you can see the current values of these (and many other) parameters. To change one, use `sysctl -w`, as in `sysctl -w net.inet.tcp.keeppidle=28800`.

6.6.2 - How can I turn on directed broadcasts?

Normally, you don't want to do this. This allows someone to send traffic to the broadcast address(es) of your connected network(s) if you are using your OpenBSD box as a router.

There are some instances, in closed networks, where this may be useful, particularly when using older implementations of the NetBIOS protocol. This is another `sysctl`. `sysctl -w net.inet.ip.directed-broadcast=1` turns this on. Read about [smurf attacks](#) if you want to know why it is off by default.

6.6.3 - I don't want the kernel to dynamically allocate a certain port

There is a `sysctl` for this also. From [sysctl\(8\)](#):

Set the list of reserved TCP ports that should not be allocated by the kernel dynamically. This can be used to keep daemons from stealing a specific port that another program needs to function. List elements may be separated by commas and/or whitespace.

```
# sysctl -w net.inet.tcp.baddynamic=749,750,751,760,761,871
```

It is also possible to add or remove ports from the current list.

```
# sysctl -w net.inet.tcp.baddynamic=+748
# sysctl -w 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.

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 on who can reach your host will be able to mount your NFS exports.

NFS depends upon [portmap\(8\)](#) to be running before it will operate. `portmap(8)` is now off by default on OpenBSD 3.2 and later, so you must enable it in [rc.conf\(8\)](#) by changing the `portmap` line to read:

```
portmap=YES
```

and reboot to make it take effect.

The setup consists of a server with the ip **10.0.0.1**. This server will be serving NFS only to clients within that network. The first step to setting up NFS is to setup your `/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 your `/etc/exports` file, and it is best that you read the [exports\(5\)](#) man page. For this example we have an `/etc/exports` 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. `-alldirs` specifies that clients will be able to mount at any point under the `/work` mount point. `-ro` specifies that it will only be allowed to be mounted read-only. 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.

Once your `/etc/exports` file is setup, you can go ahead and setup your NFS server. You should first make sure that options `NFSSERVER` & `NFSCIENT` are in your kernel configuration. (GENERIC kernel has these options included.) Next, you should set `nfs_server=YES` in `/etc/rc.conf`. This will bring up both `nfsd(8)` and `mountd(8)` when you reboot. Now, you can go ahead and start the daemons yourself. These daemons need to be started as root, and you need to make sure that `portmap(8)` is running on your system. Here is an example of starting `nfsd(8)` which serves on both TCP and UDP using 4 daemons. You should set an appropriate

number of NFS server daemons to handle the maximum number of concurrent client requests that you want to service.

```
# /sbin/nfsd -tun 4
```

Not only do you have to start the nfsd(8) server, but you need to start mountd(8). This is the daemon that actually services the mount requests on NFS. To start mountd(8), make sure an empty mountdtab file exists, and run the daemon:

```
# echo -n >/var/db/mountdtab
# /sbin/mountd
```

If you make changes to /etc/exports while NFS is already running, you need to make mountd aware of this! Just HUP it:

```
# kill -HUP `cat /var/run/mountd.pid`
```

Checking Stats on NFS

From here, you can check to make sure that all these daemons are up and 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(8) 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
```

Mounting NFS Filesystems

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 your system mount upon boot, add something like this to your /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. Such as:

```
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.

6.8 - Domain Name Service - DNS, BIND, and named

6.8.1 What is DNS?

Domain Name Service is a network facility allowing IP network domains to provide name-to-IP address resolution and IP address-to-name resolution in response to a query. Your OpenBSD installation is configured by default as a DNS client but not as a DNS server. That is, your OpenBSD installation can perform a DNS query against a domain name server for the address of a machine, but it cannot answer such DNS queries itself unless you specifically configure it to do so.

My OpenBSD machine is currently connected to the Internet via my ISP, so I can use the [nslookup\(8\)](#) utility to execute the DNS query:

```
$ nslookup www.openbsd.org
Server: ns4.us.prserv.net
Address: 165.87.201.244

Non-authoritative answer:
Name: www.openbsd.org
Address: 129.128.5.191
```

165.87.201.244 is the name server which answered, because it is the nameserver that my ISP told me to use with my account and whose number is entered in [/etc/resolv.conf](#). But the answer was not authoritative. For an authoritative answer, let's find which is the authoritative DNS server for the *openbsd.org* domain and ask it for the address of *www.openbsd.org*:

```
# Identify the name servers for openbsd.org
# with the help of my ISP's name server.
$ nslookup -type=NS openbsd.org
Server: ns4.us.prserv.net
Address: 165.87.201.244

Non-authoritative answer:
openbsd.org nameserver = cvs.openbsd.org
openbsd.org nameserver = gandalf.sigmasoft.com
openbsd.org nameserver = cs.colorado.edu
openbsd.org nameserver = ns.appli.se
openbsd.org nameserver = zeus.theos.com

Authoritative answers can be found from:
cvs.openbsd.org internet address = 199.185.137.3
gandalf.sigmasoft.com internet address = 198.144.202.98
cs.colorado.edu internet address = 128.138.243.151
ns.appli.se internet address = 194.198.196.230
zeus.theos.com internet address = 199.185.137.1

# Use the info gained to query for an authoritative
# resolution: query the authoritative zeus.theos.com.
$ nslookup www.openbsd.org zeus.theos.com
Server: zeus.theos.com
Address: 199.185.137.1

Name: www.openbsd.org
Address: 129.128.5.191
```

zeus.theos.com is, one would suppose, running OpenBSD and is properly configured to be a DNS server for the *openbsd.org* domain.

6.8.1.1 Where can I learn all about DNS and its implementation under OpenBSD?

- See RFCs [1033](#), [1034](#), and [1035](#) for more information on the Internet name-domain system.
- Read the O'Reilly Associates book [DNS and BIND](#).
- Read the [OpenBSD Manual](#) especially the pages for
 - [dig\(1\)](#)
 - [nslookup\(8\)](#)
 - [gethostbyname\(3\)](#)
 - [named\(8\)](#)
 - [resolver\(3\)](#)
 - [resolver\(5\)](#)

The [dig\(1\)](#) command is especially useful, because it can query a domain and return information in much the same record format as required in BIND configuration files. You can use [dig\(1\)](#) to examine name servers you know to be operating properly as a way of comparing your setup to theirs.

6.8.2 Does my machine need to be a domain name server?

If you aren't sure that you need your machine to perform the role of DNS server, don't configure it as one. The OpenBSD installation does not, by default, activate your machine as a domain name server, though all necessary files are installed. For most workstations, just the [/etc/hosts](#) file naming local machines' IP addresses and the

[/etc/resolv.conf](#) file for indicating which DNS servers serve you out on the intranet or internet is sufficient.

On the other hand, you might need to set up a machine as a domain name server:

- If you have an IP LAN on which you do not wish to replicate "hosts" files of local addresses machine by machine. In such a case, you may configure your OpenBSD machine as a DNS server and serve queries from the other machines on your LAN.
 - **Note:** There is no practical restriction on the number of DNS servers on a LAN. Any or all machines on the LAN may offer DNS service if they are so configured. Whether any such server is considered authoritative from outside your LAN (or is even known from outside your LAN) is a configuration factor that typically is controlled at the next level up from your LAN in the domain hierarchy.
- If you have an IP LAN on which reside machines you will wish to be findable via DNS query by machines on another IP LAN or WAN.
- If you experience difficulties resolving the local hostname to an IP address, or resolving other local names to IP addresses even though you have correct */etc/hosts* and */etc/resolv.conf* (E.g., Netscape on OpenBSD sometimes exhibits this behavior because it implements its own resolver instead of just using *gethostbyname(3)* to look up addresses.)

One more consideration is speed of execution. Since name resolution is an iterative process, in which the name server makes repeated queries to other nameservers for addresses in remote domains, name resolution may take slightly longer if you have a modem connection to the Internet and are querying your own DNS server for remote addresses (which will then iteratively query remote name servers via the modem) than if you are querying your ISP's name server (which probably has a faster connection to remote name servers).

6.8.3 What are the software components of the DNS server?

- `named` ("*name daemon*")
- Configuration files in the directory hierarchy under */var/named/*

6.8.3.1 What level of BIND is supported?

BIND is the name of the behavioral specification of a domain name server. Domain name server components exist to collectively implement BIND.

There are three distinct BIND specifications:

1. BIND 4
2. BIND 8
3. BIND 9

As installed, OpenBSD `named` supports BIND 4.x.

6.8.3.2 What are some of the alternatives to providing DNS via the default BIND 4.x implementation?

- The BIND 9.x implementation in */usr/ports/net/bind9*. (See [ports](#))

6.8.3.2.1 Security note

If you use these alternative implementations of domain name service, you are providing a critical network service using software which may not have been subject to quite the same level of scrutiny as the [security-audited](#) `named` name daemon in the base installation. This is a significant consideration, since if a domain name server is compromised, resolvers using that name server can be re-directed to impostor sites.

6.8.4 How much do I have to install?

If the default networking setup was installed properly at OpenBSD installation time, everything is already installed. You just have to configure the name daemon ("`named`").

6.8.5 How do I configure DNS?

You configure OpenBSD DNS by editing and/or creating files that control the name daemon `named`. These files reside by default in the directory */var/named* and its subdirectories, especially the file */var/named/named.boot* which is the initialization file for `named`. There are also a couple of other configuration steps to be taken in */etc*.

In this document, we will configure the name daemon on *nemo.yewtopia.com* be the primary nameserver for the (very small!) domain *yewtopia.com*. The address of *nemo.yewtopia.com* is *192.168.1.9*. Two other machines are on that subnet, *crater.yewtopia.com* at *192.168.1.1* and *earhart.yewtopia.com* at *192.168.1.2*.

6.8.5.1 Configuration in */var/named*

6.8.5.1.1 /var/named/named.boot

```

; tell what subdir has the lookup database files
directory      /namedb

; type      domain      source host/file backup file
cache       .           root.cache
primary     0.0.127.IN-ADDR.ARPA localhost.rev

; example primary server config:
primary     yewtopia.com yewtopia
primary     1.168.192.IN-ADDR.ARPA yewtopia.rev

```

This tells the initialization process in what subdirectory and under which filenames to find the configuration files for *yewtopia.com*.

6.8.5.1.2 /var/named/namedb/localhost.rev

```

; Reverse lookup for localhost interface
@           IN          SOA      nemo.yewtopia.com.  your_id.nemo.yewtopia.com.  (
                                           14          ; Serial
                                           3600       ; Refresh
                                           900        ; Retry
                                           3600000    ; Expire
                                           3600      ) ; Minimum
1           IN          NS       nemo.yewtopia.com.
1           IN          PTR      localhost.yewtopia.com.

```

6.8.5.1.3 /var/named/namedb/yewtopia

```

; yewtopia.com domain database
@           IN          SOA      nemo.yewtopia.com.  your_id.nemo.yewtopia.com.  (
                                           14          ; Serial
                                           3600       ; Refresh
                                           900        ; Retry
                                           3600000    ; Expire
                                           3600      ) ; Minimum
           IN          NS       nemo.yewtopia.com.

; Addresses
localhost.yewtopia.com.  IN A    127.0.0.1
crater.yewtopia.com.    IN A    192.168.1.1
earhart.yewtopia.com.   IN A    192.168.1.2
nemo.yewtopia.com.      IN A    192.168.1.9

```

6.8.5.1.4 /var/named/namedb/yewtopia.rev

```

; yewtopia domain reverse lookup database
@           IN          SOA      nemo.yewtopia.com.  your_id.nemo.yewtopia.com.  (
                                           14          ; Serial
                                           3600       ; Refresh
                                           900        ; Retry
                                           3600000    ; Expire
                                           3600      ) ; Minimum
1.168.192.in-addr.arpa. IN      NS       nemo.yewtopia.com.

; Addresses
1.1.168.192.in-addr.arpa. IN PTR  crater.yewtopia.com.
2.1.168.192.in-addr.arpa. IN PTR  earhart.yewtopia.com.
9.1.168.192.in-addr.arpa. IN PTR  nemo.yewtopia.com.

```

6.8.5.2 Configuration in /etc**6.8.5.2.1 /etc/resolv.conf**

Make sure */etc/resolv.conf* now points to the domain of the local machine (instead of, for example, your ISP's name server) so that name resolution requests actually get sent to the **named** you have configured!

```

domain yewtopia.com
lookup file bind

```


6.8.5.2.2 /etc/hosts

If you previously had added the addresses of various machines to the `/etc/hosts` file, you might consider shortening your `/etc/hosts` file back to the default:

```
# Host addresses
127.0.0.1      localhost      localhost.localdomain
192.168.1.9    nemo           nemo.yewtopia.com
```

So that **named** isn't bypassed in favor of (possibly outdated) addresses in the `/etc/hosts` file. **Make sure you have at least the default `localhost` entry** or your network won't start properly! Note also `nemo` must appear in its own hosts file or you will see a (mostly harmless) error message at bootup when `/etc/netstart` invokes [route\(8\)](#) in order to add `nemo` (whose name appears in `/etc/myname`).

6.8.5.3 Using [dig\(1\)](#) to examine the results.

```
$ dig @nemo.yewtopia.com yewtopia.com any any

; <<>> DiG 2.2 <<>> @nemo.yewtopia yewtopia any any
; (1 server found)
;; res options: init recurs defnam dnsrch
;; got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 59904
;; flags: qr rd ra; Ques: 1, Ans: 2, Auth: 0, Addit: 1
;; QUESTIONS:
;;   yewtopia.com, type = ANY, class = ANY

;; ANSWERS:
yewtopia.com.  3600    SOA     nemo.yewtopia.com.  your_id.nemo.yewtopia.com. (
                14      ; serial
                3600   ; refresh (1 hour)
                900   ; retry (15 mins)
                3600000 ; expire (41 days 16 hours)
                3600 ) ; minimum (1 hour)
yewtopia.com.  3600    NS      nemo.yewtopia.com.

;; ADDITIONAL RECORDS:
nemo.yewtopia.com.  3600    A       192.168.1.9

;; Total query time: 4 msec
;; FROM: nemo to SERVER: nemo.yewtopia.com 192.168.1.9
;; WHEN: Tue May  2 23:47:19 2000
;; MSG SIZE sent: 25 rcvd: 102
```

6.8.6 How and when do I start and stop DNS?

6.8.6.1 Starting DNS

The name daemon **named** is launched during system startup from `/etc/rc` if the line installed by default in `/etc/rc.conf`.

```
named_flags=NO      # for normal use: ""
```

is changed to

```
named_flags=""      # for normal use: ""
```

Also, examine these lines in `/etc/rc.conf`:

```
named_user=named      # Named should not run as root unless necessary
named_chroot=/var/named # Where to chroot named if not empty
```

These defaults will be correct for nearly all setups.

To start **named** by hand, use the [ndc\(8\)](#) command. For example:

```
# ndc start
      or
# ndc restart
```

6.8.6.2 Stopping DNS

The best way to stop the name daemon is to use the [ndc\(8\)](#) command. For example:

```
# ndc stop
```

If this fails to work, find the process id of named and use the [kill\(1\)](#) command to end that process. The PID for **named** while it is running is found as the first line in the file `/var/named/named.pid`:

```
# cat /var/named/named.pid
4608
named -t /var/named -u named
# kill -KILL 4608
```

6.8.6.3 Restarting DNS with an altered configuration

To cause a running instance of the name daemon to restart itself reloading its configuration after you have made changes, send it a "hangup" signal:

```
# kill -HUP 4608
```

or by using the [ndc\(8\)](#) command. For example:

```
# ndc reload
```

6.8.7 How do I block AXFR queries?

example:

```
garden:/home/jeremy$ host -l openssh.com
openssh.com.      NS      zeus.theos.com.
openssh.com.      NS      cvs.openbsd.org.
openssh.com.      NS      gandalf.sigmasoft.com.
openssh.com.      NS      cs.colorado.edu.
openssh.com.      NS      ns.appli.se.
openssh.com.      A      199.185.137.4
cvs.openssh.com.  A      199.185.137.4
localhost.openssh.com. A      127.0.0.1
```

This information is useful for debugging DNS, but in some instances you may not want this output publicly offered. If you are using classless in-addr(rfc2317) for reverse, `host -l` may report every domain that your system hosts! This can easily be remedied by the 'allow-transfer' clause in your zone file.

If you're using Bind8 you need to specify the hosts you want to allow to transfer zones in your individual zone file(s):

```
zone "foo.com" in {
    type master;
    file "directory/zonefile";
    allow-transfer {
        127.0.0.1;
        10.0.0.6;
        10.0.255.12;
    };
};
```

You can also block transfers for all domains by editing `/var/named.conf` and add the 'allow-transfer' parameter to the 'options' section of the configuration file:

```
options {
    allow-transfer { 127.0.0.1; };
};
```

The Bind8 method also works with Bind9.

If you are using Bind 4 (default in OpenBSD) you can edit `/var/named/named.boot` and use the 'xfrnets' option.

```
xfrnets 209.142.221.5 12.7.96.7
```

```

; type      domain                source host/file                backup file
cache      .                        .                                root.cache
primary    0.0.127.IN-ADDR.ARPA    localhost.rev

```

Bind 4 allows transfers from entire classes so it is not as exact. Typically, the only hosts that need to perform transfers are your DNS slaves and hosts you may want to debug from (127.0.0.1 is usually a good host to allow transfers from!) Blocking AXFR queries adds an extra level of privacy, but can hinder useful DNS debugging. (Thanks to [Nicholas Tang](#) for that tip.)

6.8.8 What didn't you tell me about setting up DNS?

There's a lot we didn't tell you, for example, how to set up DNS so that queries for intranet domains that aren't visible from the root of the domain hierarchy get relayed to servers within your enterprise. Read the [documents we recommended](#) for more information on DNS.

6.9 - Setting up a PPTP connection in OpenBSD

NOTE: This does not apply to **ALL** ADSL providers, but much information can be gleaned from the setup here. This is known to work for [Inode](#), an ADSL provider in Austria.

To start off, you need to install pptp. The port is located at `/usr/ports/net/pptp`. Read [FAQ 8, Ports](#) for more information on the OpenBSD ports tree.

Because of a conflict between the In-Kernel [gre\(4\)](#) support and pptp, you will need to re-compile your kernel, removing support for gre(4).

Patch to remove GRE(4) support.

```

Index: GENERIC
=====
RCS file: /cvs/src/sys/conf/GENERIC,v
retrieving revision 1.86
diff -u -r1.86 GENERIC
--- GENERIC      14 Mar 2002 00:42:25 -0000      1.86
+++ GENERIC      17 May 2002 01:52:17 -0000
@@ -87,7 +87,7 @@
 pseudo-device  enc      1      # option IPSEC needs the encapsulation interface
 pseudo-device  bridge  2      # network bridging support
 pseudo-device  vlan    2      # IEEE 802.1Q VLAN
-pseudo-device  gre      1      # GRE encapsulation interface
+#pseudo-device gre      1      # GRE encapsulation interface
 #pseudo-device strip  1      # Starmode Radio IP interface

 pseudo-device  pty      64     # pseudo-terminals

```

To recompile your kernel, check out OpenBSD source via cvs (refer to [AnonCVS](#) web page for more information), apply the following patch, and rebuild your kernel as per [FAQ 5, Building a kernel](#).

After you have the **pptp** package installed and a new kernel, you need to edit a few files to setup for your connection. This packages uses the in-house OpenBSD [ppp\(8\)](#), so if you are familiar with ppp(8), much of the setup is the same. Also, refer to [FAQ 6, PPP](#).

- 1 - `/etc/ppp/options`
- 2 - `/etc/ppp/pap-secrets`

For the `/etc/ppp/options` file, a setup like below will most likely do all that you need:

```

# cat /etc/ppp/options
name "LOGINNAME"
noauth
noipdefault
defaultroute
debug

```

LOGINNAME should be replaced with your User-ID.

The `/etc/ppp/pap-secrets` a line like:

```

# cat /etc/ppp/pap-secrets

```

```
LOGINNAME 10.0.0.138 PASSWORD
```

Where LOGINNAME is your User-ID and PASSWORD is your password. 10.0.0.138 is the IP assigned to your MODEM in the case that you are using ADSL, etc. Make sure this file stays readonly by root (mode 600).

6.9.1 - Assigning an address to your Network Interface

In the above example, our modem came with a preconfigured interface of 10.0.0.138. We now need to assign an address to OUR interface. It's best to pick an IP close to the one given by your MODEM, or use the static IP assigned to you. Read more about setting up interfaces in [FAQ 6, Setup](#).

Once your interface is setup, you should be able to create a pptp connection with the command:

```
# /usr/local/sbin/pptp 10.0.0.138 &
```

Since this uses the in-house OpenBSD ppp(8), two processes are started. You can kill pptp by killing both these processes:

```
# kill -9 [pid of pppd]
$ kill -9 [pid of pptp]
```

It is recommended to open `/var/log/messages` in a extra terminal window, to recognize possible problems.

```
# tail -f /var/log/messages
```

We also suggest that you put the startup command in `/etc/rc.local` so that you automatically connect on reboot.

6.10 - 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 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 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 [bridgename.bridge0](#). Here is an example for my situation here:

```
$ cat /etc/bridgename.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 [brconfig\(8\)](#) 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 [brconfig\(8\)](#) or in [bridgename.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.

[\[FAQ Index\]](#) [\[To Section 5 - Kernel configuration and Disk Setup\]](#) [\[To Section 7 - Keyboard controls\]](#)



www@openbsd.org

\$OpenBSD: faq6.html,v 1.168 2003/04/04 17:49:08 nick Exp \$



[\[FAQ Index\]](#) [\[To Section 6 - Networking\]](#) [\[To Section 8 - General Questions\]](#)

7 - Keyboard and Display Controls

Table of Contents

- [7.1 - How do I remap the keyboard? \(wscons\)](#)
 - [7.2 - Is there gpm or the like in OpenBSD?](#)
 - [7.3 - How do I clear the console each time a user logs out?](#)
 - [7.4 - Accessing the console scrollbar buffer. \(alpha/macppc/i386\)](#)
 - [7.5 - How do I switch consoles? \(i386\)](#)
 - [7.6 - How can I use a console resolution of 80x50? \(i386\)](#)
 - [7.7 - How do I use a serial console?](#)
 - [7.8 - How do I blank my console? \(wscons\)](#)
-

7.1 - How do I remap the keyboard? (*wscons*)

The ports that use the [wscons\(4\)](#) console driver: [alpha](#), [i386](#), and [macppc](#).

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 -w keyboard.encoding=uk
```

In the next example, we will remap "Caps Lock" to be "Control L":

```
# wsconsctl -w keyboard.map+="keysym Caps_Lock = Control_L"
```

7.2 - Is there gpm or the like in OpenBSD?

For the [alpha](#) and [i386](#) platforms, OpenBSD provides [wsmoused\(8\)](#), a port of FreeBSD's [moused\(8\)](#). It can be enabled automatically at startup by editing the appropriate line in [rc.conf\(8\)](#).

7.3 - Clearing the console each time a user logs out.

To do this you must add a line in [/etc/gettytab\(5\)](#). Change the current section:

```
P|Pc|Pc console:\
:np:sp#9600:
```

adding the line ":cl=\E[H\E[2J:" at the end, so that it ends up looking like this:

```
P|Pc|Pc console:\
:np:sp#9600:\
:cl=\E[H\E[2J:
```

7.4 - Accessing the Console Scrollback Buffer (*alpha/macppc/i386*)

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]+[PGUP] and [SHIFT]+[PGDN].

The default scrollback buffer, or the number of pages that you can move up and view, is 8.

7.5 - How do I switch consoles? (*i386*)

The user on an i386 system can switch between five virtual terminals from the console by hitting [CTRL]+[ALT]+[F1], [CTRL]+[ALT]+[F2], [CTRL]+[ALT]+[F3], [CTRL]+[ALT]+[F4] and [CTRL]+[ALT]+[F6].

This can also be done from within the X environment, which will take you out of the graphical environment back to a text-mode environment. To return to the graphical environment, hit [CTRL]+[ALT]+[F5].

7.6 - How do I use a console resolution of 80x50? (*i386*)

i386 users normally get a console screen of 25 lines of 80 characters. However, the 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 vertical resolution, 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/vt2201.808
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 the resolution of 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 accessible through [CTRL] - [ALT] - [F6].

Note that [CTRL] - [ALT] - [F1] is screen 0 here. If you wish to alter the 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. You can also add more virtual terminals to your OpenBSD system by altering [/etc/ttys\(5\)](#), which can then be made whichever resolution you prefer.

As one might expect, all these commands can also be entered at the command prompt, as root.

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.

7.7 - 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 to 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:

```
tty00    "/usr/libexec/getty std.9600"    vt100    on secure
```


Here, `tty00` is the serial port we are using as a console. The "on" activates the [getty](#) for that serial port so it can 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. 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.

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
```

assuming you want the first serial port to be used 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.

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 i386 systems are capable of using a serial port as the console device. Your results may vary -- this port may not be available to the operating system once booted, thus requiring monitoring using TWO serial ports.
- 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 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 two serial ports on a SPARC are labeled `ttya` and `ttyb`.
- 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 may be able to use the keyboard and monitor, as well, though some combinations of SPARC hardware have trouble with the keyboard when doing this.

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](#).

7.8 - 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 -w 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".

[\[FAQ Index\]](#) [\[To Section 6 - Networking\]](#) [\[To Section 8 - General Questions\]](#)



www@openbsd.org

\$OpenBSD: faq7.html,v 1.49 2003/04/04 17:07:54 nick Exp \$



[\[FAQ Index\]](#) [\[To Section 7 - Keyboard controls\]](#) [\[To Section 9 - Tips for linux users\]](#)

8 - General Questions

Table of Contents

- [8.2 - How do I change virtual terminals? \(I386 ONLY\)](#)
 - [8.3 - I forgot my root password..... What do I do!](#)
 - [8.4 - X won't start, I get lots of error messages](#)
 - [8.5 - What is CVS, and how do I use it?](#)
 - [8.6 - What is the ports tree?](#)
 - [8.7 - What are packages?](#)
 - [8.8 - Is there any way to use my floppy drive if it's not attached during boot?](#)
 - [8.9 - OpenBSD Bootloader \(i386 specific\)](#)
 - [8.10 - Using S/Key on your OpenBSD system](#)
 - [8.11 - Why is my Macintosh losing so much time?](#)
 - [8.12 - Does OpenBSD support SMP?](#)
 - [8.13 - I sometimes get Input/output error when trying to use my tty devices](#)
 - [8.14 - What web browsers are available for OpenBSD?](#)
 - [8.15 - How do I use the mg editor?](#)
 - [8.16 - Ksh does not appear to read my .profile!](#)
 - [8.17 - Why does my /etc/motd file get written over when I modified it?](#)
 - [8.18 - Why does www.openbsd.org run on Solaris?](#)
 - [8.19 - I'm having problems with PCI devices being detected](#)
 - [8.20 - Antialiased and TrueType fonts in XFree86](#)
 - [8.21 - Does OpenBSD support any journaling filesystems?](#)
 - [8.22 - Reverse DNS or Why is it taking so long for me to log in?](#)
 - [8.23 - Why do the OpenBSD web pages not conform to HTML4/XHTML?](#)
 - [8.24 - Why is my clock off by twenty-some seconds?](#)
-

8.2 - How do I change virtual terminals?

See [this article](#).

8.3 - I forgot my root password, what do I do now?

A few steps to recovery

1. Boot into single user mode. For i386 arch type boot -s at the boot prompt.
2. mount the drives.
`fsck -p / && mount -uw /`
3. If /usr is not the same partition that / is (and it shouldn't be) then you will need to mount it, also
`fsck -p /usr && mount /usr`
4. run `passwd(1)`
5. boot into multiuser mode.. and *remember* your password!

8.4 - X won't start, I get lots of error messages

If you have X completely set up and you are using an XF86Config that you know works then the problem most likely lies in the machdep.allowaperture. You also need to make sure that:

```
option APERTURE
```

is in your kernel configuration. [It is already in the GENERIC kernel]

Then you need to edit `/etc/sysctl.conf` and set `machdep.allowaperture=2`. This will allow X to access the aperture driver. This would already be set if you said that you would be running X when asked during the install. OpenBSD requires for all X servers that the aperture driver be set, because it controls access to the I/O ports on video boards.

For other X problems on the i386, consult the XFree86 Online Documentation at <http://www.xfree86.org/support.html>.

8.5 - What is CVS? and How do I use it?

CVS is what the OpenBSD project uses to control changes to the source code. CVS stands for Concurrent Versions System. You can read more about CVS at <http://www.cvshome.org/>. CVS can be used by the end user to keep up to date with source changes, and changes in the ports tree. CVS makes it extremely simple to download the source via one of the many CVS mirrors for the project.

How to initially setup your CVS environment

You can retrieve the sources from one of the OpenBSD AnonCVS servers. These servers are listed on <http://www.openbsd.org/anoncv.html>. Once you have chosen a server you need to choose which module you are going to retrieve. There are three main modules available for checkout from the CVS tree. These are:

- *src* - The src module has the complete source code for OpenBSD. This includes userland and kernel sources.
- *ports* - The ports module holds all you need to have the complete OpenBSD ports tree. To read more on the OpenBSD ports tree, read [FAQ 8, Ports](#) of the OpenBSD FAQ.
- *XF4* - The XF4 module contains the source to compile XFree 4.

Now that you have decided which module that you wish to retrieve, there is one more step left before you can retrieve it. You must decide which method to use. CVS by default retrieves files using [ssh\(1\)](#), but some AnonCVS servers allow for the use of [rsh](#). For those of you behind a firewall there are also the options of `pserver` and some AnonCVS servers run ssh on port 2022. Be sure to check <http://www.openbsd.org/anoncv.html> for which servers support what protocols. Next I will show how to do a simple source checkout. Here I will be using an AnonCVS server located in the U.S., but remember that if you are outside of the U.S you need to use a server that is located nearby. There are many AnonCVS servers located throughout the world, so choose one nearest you. I will also be using ssh to retrieve the files.

```
$ export CVSROOT=anoncv@anoncv.usa.openbsd.org:/cvs
$ cvs get src
Warning: Remote host denied X11 forwarding, perhaps xauth program could not be run on the server side.
cvs checkout: in directory src:
cvs checkout: cannot open CVS/Entries for reading: No such file or directory
cvs server: Updating src
U src/Makefile
[snip]
```

Notice here also that I set the CVSROOT environment variable. This is the variable that tells [cvs\(1\)](#) which AnonCVS server to use. This can also be specified using the `-d` option. For example:

```
$ cvs -d anoncv@anoncv.usa.openbsd.org:/cvs get src
```

These commands should be run in `/usr`, which will then create the directories of `/usr/src`, `/usr/ports`, and `/usr/www`. Depending, of course, on which module you checkout. You can download these modules to anywhere, but if you wanted to do work with them (ie make build), it is expected that they be at the place above.

Keeping your CVS tree up-to-date

Once you have your initial tree setup, keeping it up-to-date is the easy part. You can update your tree at any time you choose, some AnonCVS servers update more often than others, so again check <http://www.openbsd.org/anoncv.html>. In this example I will be updating my `www` module from `anoncv.usa.openbsd.org`. Notice the `-q` option that I use, this makes the output not so verbose coming from the server.

```
$ echo $CVSROOT
anoncv@anoncv.usa.openbsd.org:/cvs
$ cvs -q up -Pd www
Warning: Remote host denied X11 forwarding, perhaps xauth program could not be run on the server side.
U www/want.html
M www/faq/faq8.html
ericj@oshibana:~>
```

Other cvs options

For some, bandwidth and time are serious problems when updating repositories such as these. So CVS has a `-z[1-9]` option which uses `gzip` to compress the data. To use it, do `-z[compression-level]`, for instance, `-z3` for a compression level of 3.

8.6 - What is the ports tree?

The ports tree is a set of Makefiles that download, patch, configure and install userland programs so you can run them in OpenBSD environment without having to do all that by hand. You can get the ports tree from any of the OpenBSD FTP servers in `/pub/OpenBSD/3.2/ports.tar.gz`. The most recent ports are available via the 'ports' cvs tree, or `/pub/OpenBSD/snapshots/ports.tar.gz`. For most of you however, packages will be a much better option. Packages are created from ports and are already compiled and ready to use. To read more on packages read [FAQ 8, Packages](#).

Important note about keeping your system and ports in sync

OpenBSD has three "active" versions at any point in time:

- [Release](#): What is on the CD.
- [Stable](#): Release, plus security and reliability enhancements
- [Current](#): The development version of OpenBSD.

DO NOT mix versions of Ports and OpenBSD!

If your system is Release, use the Release version of the ports tree. Don't try to use a -Current version of the Ports tree on a -Release or -Stable system. Not only is it not likely to work, you will irritate people when you ask for help about why "nothing seems to work!" Note that there is a [-Stable](#) branch of the Ports tree as well, where critical fixes to -Release ports will be made.

Yes, this really does mean a wonderful new port will not typically work on your "older" system -- even if that system was -current just a few weeks ago.

If you do not have the ports tree installed, you can download it via any of OpenBSD's [FTP servers](#), or of course, from the [CDROM](#). The file is `ports.tar.gz`, and you want to untar this in the `/usr` directory, which will create `/usr/ports`, and all the directories under it. For example:

```
$ ftp ftp://ftp.openbsd.org/pub/OpenBSD/3.2/ports.tar.gz
$ sudo cp ports.tar.gz /usr
$ cd /usr; sudo tar xzf ports.tar.gz
```

A snapshot of the ports tree is also created daily and can be downloaded from any of the [OpenBSD FTP servers](#) as `/pub/OpenBSD/snapshots/ports.tar.gz`. If you are installing a snapshot of OpenBSD, you should use a matching snapshot of ports. Again, make sure you keep your ports tree and your OpenBSD system in sync.

What ports are available? and how do i find them?

Use the ports tree to search for keywords. To do this use `make search key="searchkey"`. Here is an example of a search for 'samba':

```
$ make search key="samba"
Port:      samba-2.2.5
Path:      net/samba
Info:      SMB and CIFS client and server for UNIX
Maint:     Daniel Hartmeier <daniel@benzedrine.cx>
Index:     net
L-deps:
B-deps:    devel/autoconf
R-deps:
Archs:     any

Port:      ADMsmb-0.2
Path:      /usr/ports/security/ADMsmb
Info:      Samba security scanner
Maint:     Jason Peel <jsyn@openbsd.org>
Index:     security
L-deps:
B-deps:
R-deps:
Archs:     any
```

Installing Ports

Ports are set up to be EXTREMELY easy to make and install. Here is an example install for someone wanting to install the X11 program `xfig`. You'll notice the dependencies are automatically detected and completed.

First you need to `cd` to the dir of the program you want. If you are searching for a program, you can either update your locate database, or use the search function talked about below. Once you are in the dir of the program you want, you can just type `make install`. For example.

```
$ sudo make install
===> Extracting for xfig-3.2.2
```

```

====> xfig-3.2.2 depends on shared library: jpeg.62. - /usr/local/lib/libjpeg.so.62.0 found
====> xfig-3.2.2 depends on shared library: Xaw3d.6. - not found
====> Verifying install for Xaw3d.6. in /usr/ports/x11/Xaw3d
>> Xaw3d-1.3.tar.gz doesn't seem to exist on this system.
>> Attempting to fetch from ftp://crl.dec.com/pub/X11/contrib/widgets/Xaw3d/R6.1/.
Connected to crl.dec.com.
220 crl.dec.com FTP server (Digital UNIX Version 5.60) ready.
331 Guest login ok, send ident as password.
230 Guest login ok, access restrictions apply.
Remote system type is UNIX.
Using binary mode to transfer files.
200 Type set to I.
250 CWD command successful.
250 CWD command successful.
Retrieving pub/X11/contrib/widgets/Xaw3d/R6.1/Xaw3d-1.3.tar.gz
local: Xaw3d-1.3.tar.gz remote: Xaw3d-1.3.tar.gz
227 Entering Passive Mode (192,58,206,2,5,14)
150 Opening BINARY mode data connection for Xaw3d-1.3.tar.gz (0.0.0.0,0) (290277 bytes).
100% |*****| 283 KB 00:00 ETA
226 Transfer complete.
290277 bytes received in 101.09 seconds (2.80 KB/s)
221 Goodbye.
====> Extracting for Xaw3d-1.3
/bin/mkdir -p /usr/ports/x11/Xaw3d/work/xc/lib/Xaw3d/X11/Xaw3d
cd /usr/ports/x11/Xaw3d/work/xc/lib/Xaw3d/X11/Xaw3d; ln -sf ../../*.h .
====> Patching for Xaw3d-1.3
====> Configuring for Xaw3d-1.3
mv -f Makefile Makefile.bak
imake -DUseInstalled -I/usr/X11R6/lib/X11/config
make Makefiles
[snip]

```

Using Flavors

Many of the applications in the ports tree support different install options, called *flavors*. If a port comes in multiple flavors, you can use these options simply by setting an environment variable before you compile the port. If multiple features are needed, the FLAVOR variable can be set to a space-delimited list of the supported and desired flavors. Currently, many ports have flavors that include database support, support for systems without X, or network additions like SSL and IPv6.

```

$ pwd
/usr/ports/net/mtr
$ make show=FLAVORS
no_x11
$ env FLAVOR="no_x11" make
====> mtr-0.42-curses depends on: gmake-3.79.1 - found
====> Checking files for mtr-0.42-no_x11
>> mtr-0.42.tar.gz doesn't seem to exist on this system.
>> Attempting to fetch /usr/ports/distfiles/mtr-0.42.tar.gz from ftp://ftp.bitwizard.nl/mtr/.
[snip]
$ sudo env FLAVOR="no_x11" make install
====> Faking installation for mtr-0.42-no_x11
====> Building package for mtr-0.42-no_x11
Creating package /usr/ports/packages/i386/All/mtr-0.42-no_x11.tgz
Using SrcDir value of /usr/ports/net/mtr/w-mtr-0.42-no_x11/fake-i386-curses/usr/local
Creating gzip'd tar ball in '/usr/ports/packages/i386/All/mtr-0.42-no_x11.tgz'
====> Installing mtr-0.42-no_x11 from /usr/ports/packages/i386/All/mtr-0.42-no_x11.tgz

```

Listing Installed ports/packages

You can see a list of both ports and packages by using the `pkg_info` command.

```

$ /usr/sbin/pkg_info
zsh-4.0.6          The Z shell.
screen-3.9.13     A multi-screen window manager.
emacs-21.2        GNU editing macros.
tcsh-6.12.00     An extended C-shell with many useful features.
bash-2.05b        The GNU Borne Again Shell.
zip-2.3           Create/update ZIP files compatible with pkzip.
ircII-20020403    An enhanced version of ircII, the Internet Relay Chat client
ispell-3.2.06     An interactive spelling checker.
tin-1.4.5         TIN newsreader (termcap based)
procmail-3.22     A local mail delivery agent.
strobe-1.06       Fast scatter/gather TCP port scanner
lsof-4.63         Lists information about open files.
ntp-4.1.72        Network Time Protocol Implementation.
ncftp-3.1.4       ftp replacement with advanced user interface

```

nmh-1.0.4	The New MH mail handling program
bzip2-1.0.2	A block-sorting file compressor

Other Information

More information about the ports can be found in the [ports\(7\)](#) man page and on the [Ports page](#).

Our ports tree is constantly being expanded, and if you would like to help please see: <http://www.openbsd.org/porting.html>.

8.7 - What are packages?

Packages are the precompiled binaries of some of the most used programs. They are ready for use on an OpenBSD system. Again, like the ports, packages are very easy to maintain and update. Packages are constantly being added so be sure to check each release for additional packages.

Here is a list of tools used in managing packages.

- [pkg_add\(1\)](#) - a utility for installing software package distributions
- [pkg_create\(1\)](#) - a utility for creating software package distributions
- [pkg_delete\(1\)](#) - a utility for deleting previously installed software package distributions
- [pkg_info\(1\)](#) - a utility for displaying information on software packages

Where to find packages

If you are a smart user and bought one of the [OpenBSD CD](#), then packages can be found on both CDs depending on your architecture. If you don't have an OpenBSD CD in your possession you can download packages from any of the ftp mirrors. You can get a list of mirrors <http://www.openbsd.org/ftp.html>. Packages are located at */pub/OpenBSD/3.2/packages* from there packages are broken down depending on architecture.

Installing Packages

To install packages, the utility [pkg_add\(1\)](#) is used. `pkg_add(1)` is an extremely easy utility to use, in the following two examples `pkg_add(1)` will be used to install a package. The first example will show `pkg_add(1)` installing a package that resides on a local disk, the second example will show an installation of a package via ftp. In both examples `screen-3.9.13` will be installed.

Installing via local disk

```
$ sudo pkg_add -v screen-3.9.13.tgz
Requested space: 749864 bytes, free space: 2239117312 bytes in /var/tmp/instmp.cpsHA27596
Running install with PRE-INSTALL for `screen-3.9.13'
extract: Package name is screen-3.9.13
extract: CWD to /usr/local
extract: /usr/local/bin/screen-3.9.13
extract: execute 'ln -sf screen-3.9.13 /usr/local/bin/screen'
extract: /usr/local/man/man1/screen.1
extract: /usr/local/info/screen.info
extract: execute '[ -f /usr/local/info/dir ] || sed -ne '1,/Menu:/p' /usr/share/info/dir > /usr/local/info/dir'
extract: execute 'install-info /usr/local/info/screen.info /usr/local/info/dir'
extract: /usr/local/lib/screen/screencap
extract: /usr/local/lib/screen/screenrc
extract: CWD to .
Runningmtree for `screen-3.9.13'
mtree -q -U -f +MTREE_DIRS -d -e -p /usr/local
Running install with POST-INSTALL for `screen-3.9.13'

+-----+
| The file /etc/screenrc has been created on your system.
| You may want to verify/edit its contents
|
| The file /usr/local/lib/screen/screencap contains a
| termcap like description of the screen virtual terminal.
| You may use it to update your terminal database.
| See termcap\(5\).
+-----+

Attempting to record package into `/var/db/pkg/screen'
Package `screen-3.9.13' registered in `/var/db/pkg/screen-3.9.13'
```

In this example the `-v` flag was used to give a more verbose output, this option is not needed, but is helpful for debugging and was used here to give a little more insight into what `pkg_add(1)` is actually doing. Notice however, that there are some valid messages given out mentioning `/etc/screenrc`. Messages like this will be given to you whether or

8 - General Questions

not you use the `-v` flag.

Installing via ftp

```
$ sudo pkg_add ftp://ftp.openbsd.org/pub/OpenBSD/3.2/packages/i386/screen-3.9.13.tgz
>>> ftp -o - ftp://ftp.openbsd.org/pub/OpenBSD/3.2/packages/i386/screen-3.9.13.tgz

+-----+
| The file /etc/screenrc has been created on your system.
| You may want to verify/edit its contents
|
| The file /usr/local/lib/screen/screencap contains a
| termcap like description of the screen virtual terminal.
| You may use it to update your terminal database.
| See termcap(5).
+-----+
```

In this example you can see that I installed the i386 package, you should substitute this with your architecture. Notice: Not all architectures have the same packages. Some ports don't work on certain architectures. In this example the `-v` flag wasn't used, so only NEEDED messages are shown.

Viewing and Deleting Installed Packages

The utility [pkg_info\(1\)](#) is used to view a list of packages that are already installed on your system. This is usually needed to find out the correct name of a package before you remove that package. To see what packages are installed on your system simple use:

```
$ pkg_info
mpgl23-0.59r      mpeg audio 1/2 layer 1, 2 and 3 player
nmap-3.00         port scanning large networks
ircII-20020403   enhanced version of ircII (internet relay chat)
screen-3.9.13    multi-screen window manager
unzip-5.50       extract, list & test files in a ZIP archive
ntp-4.1.72       Network Time Protocol implementation
icb-5.0.9        Internet CB - mostly-defunct chat client
```

To delete a package, simple take the proper name of the package as shown by [pkg_info\(1\)](#) and use [pkg_delete\(1\)](#) to remove the package. In the below example, the screen package is being removed. Notice that on some occasions there are instructions of extra objects 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-3.9.13

+-----+
| To completely deinstall the screen-3.9.13 package you need to perform
| this step as root:
|
|           rm -f /etc/screenrc
|
| Do not do this if you plan on re-installing screen-3.9.13
| at some future time.
+-----+
```

8.8 - Is there any way to use my floppy drive if it's not attached during boot?

Sure. You need to add "flags 0x20" at the end of the fd* entry and recompile your kernel. The line should be read:

```
fd*      at fdc? drive ? flags 0x20
```

After that you would be able to use the floppy drive all the times. It doesn't matter if you plugged it in after boot.

8.9 - Boot time Options - Using the OpenBSD bootloader

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 this by hand by typing:

```
boot> boot wd0a:/bsd
```

or

```
boot> b /bsd
```

This will work if device wd0a is configured as your root device.

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_i386\(8\)](#) man page

8.10 - S/Key

S/Key is a "one-time password" scheme. This allows for one-time passwords for use on un-secured channels. This can come in handy for those who don't have the ability to use ssh or any other encrypted channels. OpenBSD's S/Key implementation can use a variety of algorithms as the one-way hash. The following algorithms are available:

- [md4](#)
- [md5](#)
- [sha1](#)
- [rmd160](#).

Setting up S/Key - The first steps

To start off the file `/etc/skeykeys` must exist. If this file is not in existence, have the super-user create it. This can be done simply by doing:

```
# touch /etc/skeykeys
```

Once that file is in existence, you can initialize your S/Key. To do this you will have to use [skeyinit\(1\)](#). 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. Running `skeyinit(1)` over an insecure channel is completely not recommended, so this should be done over a secure channel (such as ssh) or the console. Once you have authorized yourself with your system password you will be asked for yet another password. This password is the S/Key *secret password*, and is **NOT** your system password. Your secret password must be at least 10 characters. We suggest using a memorable phrase containing several words as the secret password. Here is an example user being added.

```
$ skeyinit ericj
[Adding ericj]
Reminder - Only use this method if you are directly connected
          or have an encrypted channel.  If you are using telnet
          or rlogin, exit with no password and use skeyinit -s.
Enter secret password:
Again secret password:

ID ericj skey is otp-md5 99 oshi45820
Next login password: HAUL BUS JAKE DING HOT HOG
```

One line of particular importance in here is `ID ericj skey is otp-md5 99 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 was used to create your One-Time Password (otp).
- `99` - This is your sequence number. This is a number from 100 down to 1. Once it reaches one, another secret password must be created.
- `oshi45820` - This is your key.

But of more immediate importance is your password. Your password consists of 6 small words, combined together this is your password, spaces and all.

Actually using S/Key to login.

By now your skey has been initialized, and you have your password. You're ready to login. Here is an example session using S/Key to login. Starting with OpenBSD 3.0, S/Key logins work differently. For OpenBSD 3.0 and above you append `:skey` to your login name. For versions of OpenBSD previous to 3.0 you use `s/key` for the password at which time you are prompted for your S/Key password (the exception to this is `ftpd(8)` which will always present an S/Key challenge for S/Key-enabled user prior to OpenBSD < 3.0). The examples below assume OpenBSD 3.0 or higher.

```
$ 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 3.2 (GENERIC) #25: Thu Oct  3 19:51:53 MDT 2002
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 password after you've logged in once? Well to do this, 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? Well this is simple, use [skeyinfo\(1\)](#), and it will tell you what to use. For example here, I need to generate another 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
```

Not only is `otp-md5` a description of the hash used, it is also an alternate name for the [skey\(1\)](#) command. So, the simplest way to generate the next S/Key password is simply:

```
$ `skeyinfo -v`
Reminder - Do not use this program while logged in via telnet or rlogin.
Enter secret password:
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 to create these passwords, and creating them over an insecure connection isn't feasible, so how can you create multiple passwords at one time? Well 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 or rlogin.
Enter secret password:
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 telnet(1), ssh(1), and rlogin(1)

Using S/Key with `telnet(1)`, `ssh(1)`, or `rlogin(1)` is done in pretty much the same fashion as with `ftp`--you simply tack `":skey"` to the end of your username. Example:

```

$ telnet localhost
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.

OpenBSD/i386 (oshibana) (ttyp2)

login: ericj:skey
otp-md5 98 oshi45821
S/Key Password: SCAN OLGA BING PUB REEL COCA
Last login: Thu Oct  7 12:21:48 on ttty1 from 156.63.248.77
Warning: no Kerberos tickets issued.
OpenBSD 3.2 (GENERIC) #25: Thu Oct  3 19:51:53 MDT 2002

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.
$

```

8.11 - Why is my Macintosh losing so much time?

This is caused by a hardware bug. OpenBSD uses clock interrupts to keep track of the current time, but these interrupts have the lowest priority in Apple's architecture. So, under heavy load, (such as disk or network activity) clock interrupts will be lost and the Unix clock will not advance as it should.

MacOS gets around the time problem by always reading the hardware clock. OpenBSD only reads the hardware clock at boot time and thereafter ignores it. You may notice that, at shutdown, the kernel is not confident enough to write the Unix time back into the hardware clock because this time loss problem is well known.

The best solution is to run `ntpd` (found in the ports collection) and just deal with the occasional lossage. Sometimes the lossage is so bad that even `ntpd` is afraid to skip the time. In this case, add the `-g` option to `ntpd` in `/etc/rc.securelevel` to force tracking.

Another simpler but less precise solution is to run `rdate(8)` on a regular basis, for example by having a crontab entry for it, preferably with the `-a` option so there is no "jump" in time. Another good place to launch `rdate(8)` is in your `/etc/ppp/ppp.linkup` file if you are not permanently connected and are a PPP user.

See also: <http://www.macbsd.com/macbsd/macbsd-docs/faq/faq-3.html#ss3.17>

8.12 - Does OpenBSD support SMP? (Symmetric Multi-Processor)

Not at this time. Work is on-going, there is an [OpenBSD SMP project](#), though there is nothing usable yet, nor is there any time schedule for OpenBSD SMP support.

On most platforms, OpenBSD will run on an SMP system, but only utilizing one processor. The exception to this is the SPARC platform -- OpenBSD/sparc will not run on some multi-processor SPARC systems at all (General rule: one MBus module always works. Multiple MBus modules will work as long as they are single processor modules and [supported](#), of course). Multi-processor SPARC64 systems run as long as the base machine is [supported](#).

How can I help?

- Write code. There is lots of work to do.
- Donate hardware. The goal of the OpenBSD SMP project is to support as many platforms as possible, including [SPARC](#), [SPARC64](#), [Alpha](#), [MacPPC](#) and of course, the [i386](#). Donations of [desired hardware](#) are welcome!
- Don't offer to "test". The SMP project is not part of the core OpenBSD project right now for a reason -- it isn't ready for production use yet, or even general "testing". Informing the developers that it didn't work on your hardware without code to make it work isn't useful at this point. [Send in your dmesg](#), if a developer sees something interesting in your hardware, they can contact you for testing.

8.13 - 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.

8.14 - What web browsers are available for OpenBSD?

[Lynx](#), a text-based browser, is in the base system, and has SSL support. Other browsers in the ports tree, include (in no particular order):

Graphical (X) Browsers

- [Dillo](#) Minimal feature set, very fast and small, runs well on slower hardware.
- [Konqueror](#) Installed as part of the [KDE desktop environment](#).
- [Konqueror-embedded](#) (konq-e) Konqueror, using only the KDE libraries rather than all of KDE.
- [Netscape 4](#) For sparc and i386 only, not Open Source, no package available.
- [Opera](#) Commercial browser.
- [Amaya](#) The W3C's browser and editor.
- [Links+](#) Another fast and small graphical browser. (Also has a text-only mode)

Console (Text mode) Browsers

- [w3m](#) Has table and frame support (also has a graphical mode).
- [links](#) Has table support.

[Mozilla](#) does not currently work on OpenBSD, either natively or in emulation. See the [mail list](#) archives for details.

You will find all these in the [ports collection](#). All the above mentioned browsers are located in /usr/ports/www/ after the installation of the ports tree. Most are also available as pre-compiled [packages](#), available on the [FTP servers](#) and on the [CDROM](#). 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.

8.15 - How do I use the mg editor which is now part of OpenBSD?

Mg is a micro Emacs-style text editor. Micro means that it's small (Emacs is very large!) For the basics, read the [mg\(1\)](#) manual page and the [tutorial](#), as included with the source code. For more interesting questions (such as, "I don't have a Meta key!") check out the [Emacs FAQ](#).

Note that since mg is a small Emacs implementation, which is mostly similar to the text editor features of Emacs 17, it does not implement many of Emacs' other functionality. (Including mail and news functionality, as well as modes for Lisp, C++, Lex, Awk, Java, etc...)

8.16 - Ksh does not appear to read my .profile!

There are two possible reasons for this.

- .profile is not owned by the user This is easy to fix. For **username**,

```
# chown username ~username/.profile
```

- You are using ksh from within X Window System

Under xterm, argv[0] for ksh is not prepended with a dash. Prepending - to argv[0] will cause csh and ksh to know they should interpret their login files. (For csh that's .login, with a separate .cshrc that is always run when csh starts up. With ksh, 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 XFree86 behavior.

8.17 - Why does my /etc/motd file get written over 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.

8.18 - Why does www.openbsd.org run on Solaris?

Although none of the developers think it is particularly relevant, this question comes up frequently enough in the mailing lists that it is answered here. www.openbsd.org and the main OpenBSD ftp site are hosted at a SunSITE at the University of Alberta, Canada. These sites are hosted on a large Sun system, which has access to lots of storage space and Internet bandwidth. The presence of the SunSITE gives the OpenBSD group access to this bandwidth. This is why the main site runs here. Many of the OpenBSD mirror sites run OpenBSD, but since they do not have guaranteed access to this large amount of bandwidth, the group has chosen to run the main site at the University of Alberta SunSITE.

8.19 - I'm having problems with my PCI devices being detected

There exists a condition where some machines might not detect some PCI devices properly, or might freeze while detecting multiple NIC's in one machine. This is the fault of PCIBIOS, and involves a simple workaround to make it work properly. Simply enter the boot time configuration and disable PCIBIOS. An example is below:

```
boot> boot -c
Copyright (c) 1982, 1986, 1989, 1991, 1993
    The Regents of the University of California. All rights reserved.
Copyright (c) 1995-2002 OpenBSD. All rights reserved. http://www.OpenBSD.org

OpenBSD 3.2 (GENERIC) #25: Thu Oct  3 19:51:53 MDT 2002
  deraadt@i386.openbsd.org:/usr/src/sys/arch/i386/compile/RAMDISK_CD
cpu0: Intel Pentium III (Coppermine) ("GenuineIntel" 686-class, 128KB L2 cache)
1 GHz
cpu0: FPU,V86,DE,PSE,TSC,MSR,PAE,MCE,CX8,SYS,MTRR,PGE,MCA,CMOV,PAT,PSE36,MMX,FXS
R,SIMD
real mem = 267956224 (261676K)
avail mem = 243347456 (237644K)
using 3296 buffers containing 13500416 bytes (13184K) of memory
User Kernel Config
UKC> disable pcibios
UKC> quit
[... snip ...]
```

Once this is done, you can follow the directions in [FAQ 5, Building a Kernel](#) to create a new kernel so that you don't have to worry about this in the future.

8.20 - Antialiased and TrueType fonts in XFree86

See [this document](#).

8.21 - Does OpenBSD support any journaling filesystems?

No it doesn't. We use a different mechanism to achieve similar results that is called [Soft Updates](#). Please read in FAQ 14 to get more details.

8.22 - 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](#), [ftp](#), or [telnet](#). 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](#), [telnet](#), [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. Some details on the BIND v4.9.8 name server included with OpenBSD can be found [here](#).

8.23 - 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.

8.24 - 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 the cause of this discrepancy. For a more detailed description, search the web for "leap seconds UTC TAI".

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.

[\[FAQ Index\]](#) [\[To Section 7 - Keyboard controls\]](#) [\[To Section 9 - Tips for linux users\]](#)



www@openbsd.org

\$OpenBSD: faq8.html,v 1.122 2003/04/04 16:43:48 nick Exp \$



[\[FAQ Index\]](#) [\[To Section 8 - General Questions\]](#) [\[To Section 10 - System Administration\]](#)

9 - Migrating from Linux

Table of Contents

- [9.1 - Tips for Linux \(and other free Unix-like OS\) users](#)
- [9.2 - Dual boot of Linux and OpenBSD](#)
- [9.3 - Converting your Linux \(or other System-7 style\) password file to BSD-style.](#)
- [9.4 - Getting OpenBSD and Linux to interact](#)

For more information for Linux users, please refer to <http://sites.inka.de/mips/unix/bsdlinux.html>.

9.1 - Simple tips for Linux (and other free Unix-like OS) users

There are several differences between OpenBSD and Linux. These differences include but are not limited to, bootup procedure, network interface usage and disk management. Most differences are well documented, but involve searching manpages. This document tries to be an index of those differences.

- OpenBSD has a [ports tree](#). This is to accommodate the fact that at this point not many applications are native to the OpenBSD environment. This is both an attempt to get applications to work on OpenBSD for end-users and to get more applications made with OpenBSD in mind. Eventually this ports tree is used to make a nice set of binary [packages](#).
- OpenBSD uses CVS for source changes. With Linux, source code is disseminated through separate distributions. 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! There is also a very convenient and easy to use [web interface to CVS](#).
- OpenBSD periodically releases snapshots for various architectures and makes a stable, official CD release every 6 months.
- OpenBSD contains STRONG CRYPTO, which USA based OS's can't contain. (See <http://www.openbsd.org/crypto.html>) OpenBSD has also gone through heavy security auditing and many security features have already been implemented into the source tree. (IPSEC, KERBEROS).
- OpenBSD's kernel is /bsd.
- The names of hard disks are usually /dev/wd (IDE) and /dev/sd (SCSI or ATA devices emulating SCSI disks)
- /sbin/ifconfig with no arguments in Linux gives the state of all the interfaces. Under OpenBSD you need the -a flag.
- /sbin/route with no arguments in Linux gives the state of all the active routes. Under OpenBSD you need the "show" parameter, or do a **netstat -r** (nice).
- OpenBSD does NOT support Journaling Filesystems like ReiserFS, IBMs JFS or SGIs XFS. Instead we use [Soft Updates](#).
- 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) is done through pfctl (using -N). ([pfctl\(8\)](#))
 - ipfwadm is done through pfctl. ([pfctl\(8\)](#), [pf\(4\)](#), [pf.conf\(5\)](#))
 - You should look at [section 6](#) for detailed configuration assistance and information.
- Interface address is stored in [/etc/hostname.<interfacename>](#). It can be a hostname 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/sh, which is the Korn shell. Shells such as bash and tcsh can be added as packages or installed from the ports tree.
- Password management changes a lot. The main files are different. ([passwd\(1\)](#), [passwd\(5\)](#))
- 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 and Fast Etherlink XL ethernet device, etc.
- OpenBSD developers have made serious efforts to keep the manual pages up-to-date and accurate. Use the [man\(1\)](#) command to find information.

9.2 - Dual booting Linux and OpenBSD

Yes! It is possible!

Read [INSTALL.linux](#).

9.3 - Converting your Linux (or other System 7-style) password file to BSD-style

First, figure out if your Linux password file is shadowed or not. If it is, grab [John the Ripper](#) and use the unshadow utility that comes with it to merge your `passwd` and `shadow` files into one System 7-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 System 7-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 System 7-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 - Getting OpenBSD and Linux to interact

If you are migrating from Linux to OpenBSD, note that OpenBSD has `COMPAT_LINUX` enabled by default in the GENERIC kernel. 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 `redhat/base` port from your ports collection. To find out more about the Ports collection read [FAQ 8 - Ports](#). If you already have the ports tree installed, use these commands to get Linux libraries installed.

```
# cd /usr/ports/emulators/redhat/base
# make install
```

OpenBSD supports the EXT2FS file system. Use [disklabel\(8\) disk](#) (where *disk* is the device name for your disk.) to see what OpenBSD thinks your Linux partition is (but **don't** use `disklabel` or `fdisk` to make any changes to it). For further information on using `disklabel` read [FAQ 14 - Disklabel](#).

[\[FAQ Index\]](#) [\[To Section 8 - General Questions\]](#) [\[To Section 10 - System Administration\]](#)



www@openbsd.org

\$OpenBSD: faq9.html,v 1.48 2003/04/04 16:29:09 nick Exp \$



[\[FAQ Index\]](#) [\[To Section 9 - Migrating from Linux\]](#) [\[To Section 11 - Performance Tuning\]](#)

10 - System Management

Table of Contents

- [10.1 - When I try to su to root it says that I'm in the wrong group.](#)
 - [10.2 - How do I duplicate a filesystem?](#)
 - [10.3 - How do I start daemons with the system? \(Overview of rc\(8\)\)](#)
 - [10.4 - Why do users get relaying access denied when they are remotely sending mail through my OpenBSD system?](#)
 - [10.5 - I've set up POP, but I get errors when accessing my mail through POP. What can I do?](#)
 - [10.6 - Why does Sendmail ignore /etc/hosts?](#)
 - [10.7 - Setting up a Secure HTTP Server using ssl\(8\)](#)
 - [10.8 - I made changes to /etc/passwd with an editor, but the changes didn't seem to take place. Why?](#)
 - [10.9 - How do I add a user? Or delete a user?](#)
 - [10.10 - How do I create a ftp-only account?](#)
 - [10.11 - Setting up user disk quotas](#)
 - [10.12 - Setting up KerberosIV Client/Server](#)
 - [10.13 - Setting up an Anonymous FTP Server](#)
 - [10.14 - Confining users to their home directories in ftpd\(8\)](#)
 - [10.15 - Applying patches in OpenBSD](#)
 - [10.16 - Tell me about chroot\(\) Apache?](#)
 - [10.17 - I don't like the standard root shell!](#)
 - [10.18 - What else can I do with ksh?](#)
-

10.1 - Why does it say that I'm in the wrong group when I try to su root?

Existing users must be added to the "wheel" group by hand. This is done for security reasons, and you should be cautious with whom you give access to. On OpenBSD, users who are in the wheel group are allowed to use the [su\(1\)](#) userland program to become root. Users who are not in "wheel" cannot use su(1). Here is an example of a `/etc/group` entry to place the user `ericj` into the "wheel" group.

If you are adding a new user with [adduser\(8\)](#), you can put them in the wheel group by answering `wheel` at `Invite user into other groups`: This will add them to `/etc/group`, which will look something like this:

```
wheel:*:0:root,ericj
```

If you are looking for a way to allow users limited access to superuser privileges without putting them in 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 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 know what daemons should start with the system.
- `/etc/rc.conf.local` - Configuration file you can use to override settings in `/etc/rc.conf` so you don't have to touch `/etc/rc.conf` itself, which is convenient for people who upgrade often.

- `/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 should be stored.
- `/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` (or `/etc/rc.conf.local`), `/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 checked. This will always be bypassed if the file `/etc/fastboot` exists. This is certainly not a good idea though.
- Configuration Variables are read in from `/etc/rc.conf` and, afterwards, `/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
- Does various checks. (quota's, savecore, etc)
- Local daemons are run, ala `/etc/rc.local`

Starting Daemons and Services that come with OpenBSD

Most daemons and services that come with OpenBSD by default can be started on boot by simply editing the `/etc/rc.conf` configuration file. To start out take a look at the default [/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 is not to start up with the system (at least not via rc(8), read the [Anonymous FTP FAQ](#) to read more about this). In any case, each line also has a comment showing you the flags for **NORMAL** usage of that daemon or service. This doesn't mean that you must run that daemon or service with those flags. You can always use `man(1)` to see how you can have that daemon or service start up in any way you like. 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 obviously see that starting up `httpd` normally no flags are necessary. So a line like: `"httpd_flags=""` would be necessary. But to start `httpd` with `ssl` enabled. (Refer to the [SSL FAQ](#) or [ssl\(8\)](#)) You should start with a line like: `"httpd_flags="-DSSL"`.

Another approach would be to never touch `/etc/rc.conf` itself. Instead, create the file `/etc/rc.conf.local` and copy just the lines you are about to change from `/etc/rc.conf` and adjust them as you like. This may make future upgrading easier -- all the changes are in the one file.

Starting up local daemons and configuration

For other daemons that you might install with the system via ports or other ways, you will use the `/etc/rc.local` file. For example, I've installed a daemon which lies at `/usr/local/sbin/daemonx`. I want this to start at boot time. I would put an entry into `/etc/rc.local` like this:

```
if [ -x /usr/local/sbin/daemonx ]; then
    echo -n ' daemonx';      /usr/local/sbin/daemonx
fi
```

From now on, this daemon will be run 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 local daemons: daemonx.
```

rc.shutdown

`/etc/rc.shutdown` is a script that is run a 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 access denied when they are remotely sending mail through my OpenBSD system?

Try this:

```
# cat /etc/mail/sendmail.cf | grep relay-domains
```

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.html>
- <http://www.sendmail.org/antispam.html>

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.)

PS: Notice, 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 /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/>. Thawte can currently sign RSA keys for you. A procedure is being worked out to allow for DSA keys.

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/private/server.key -out /etc/ssl/server.crt
```

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 Version 7 format 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 introduces the `master.passwd` file, which has an extended format (with additional options beyond what was 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 re-create `/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\(8\)](#)

The easiest way to add a user in OpenBSD is to use the [adduser\(8\)](#) script. You can configure this to work however you like by editing `/etc/adduser.conf`. You can add users by hand via [vipw\(8\)](#), but this is the recommended way to add users. `adduser(8)` allows for consistency checks on `/etc/passwd`, `/etc/group`, and shell databases. It will create the entries and `$HOME` directories for you. It can even send a message to the user welcoming them. This can be changed to meet your needs. Here is an example user, `testuser` being added to a system. His/Her `$HOME` directory will be placed in `/home/testuser`, and given the group `guest`, and 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. I will give you the chance later to correct any input.
Enter username [a-z0-9_]: 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
Enter password []:
Enter password again []:

Name:      testuser
Password:  ****
Fullname:  Test FAQ User
Uid:       1002
Gid:       31 (guest)
Groups:    guest
HOME:      /home/testuser
Shell:     /bin/ksh
OK? (y/n) [y]: y
Added user ``testuser''
Copy files from /usr/share/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 users home directory.

```

# rmuser
Enter login name for user to remove: testuser
Matching password entry:

testuser:$2a$07$ZWnB0sbqMJ.ducQBfsTKUe3PL97Ve1AHWJ0A4uLamniLNXLLeYrEie: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 helps facilitate using these in a script.

A list of the added commands are:

- [group\(8\)](#)
- [groupadd\(8\)](#)
- [groupdel\(8\)](#)
- [groupinfo\(8\)](#)
- [groupmod\(8\)](#)
- [user\(8\)](#)
- [useradd\(8\)](#)
- [userdel\(8\)](#)
- [userinfo\(8\)](#)
- [usermod\(8\)](#)

Actually adding users

Being that [user\(8\)](#) is not interactive, the easiest way to add users efficiently is to use the [adduser\(8\)](#) command. The actual 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**, its your choice as to what you want, and doesn't change the use of the commands at all.

In this example, we are adding the same user with the same specifications as the user that was added [above](#). [useradd\(8\)](#) is much easier to use if you know the default setting before adding a user. These settings are located in */etc/usermgmt.conf* and can be viewed by doing so:

```

$ user add -D
group          users
base_dir      /home
skel_dir      /etc/skel
shell         /bin/csh
inactive      0
expire        Null (unset)
range         1000..60000

```

The above settings are what will be set unless you specify different with command line options. For example, in our case, we want the user to go to the group **guest**, not **users**. One more little hurdle with adding users, is that passwords must be specified on the commandline. This is, the encrypted passwords, so you must first use the [encrypt\(1\)](#) utility to create the password. For example: OpenBSD's passwords by default use the Blowfish algorithm for 6 rounds. Here is an example line to create an encrypted password to specify to `useradd(8)`.

```
$ encrypt -p -b 6
Enter string:
$2a$06$Y0dOZM3.4m6MObBXjeZtBOWArqC2.uRJZXUkOghbieIvSWXVJRz1q
```

Now that we have our encrypted password, we are ready to add the user.

```
# user add -p '$2a$06$Y0dOZM3.4m6MObBXjeZtBOWArqC2.uRJZXUkOghbieIvSWXVJRz1q' -u 1002 \
-s /bin/ksh -c "Test FAQ User" -m -g guest testuser
```

Note: Make sure to use `"` around the password string, not `'` 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. [adduser\(8\)](#) will give them a home dir by default of `/home/<user>`. If this is what you desire it doesn't need to be changed, however you can set this to whatever directory you wish. You can force this user to only be able to see files in their home directory by adding their username to `/etc/ftpchroot`. Using the `-A` option to [ftpd\(8\)](#), you can allow only ftpchroot logins!

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:
/: blocks 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:
/: blocks 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 - How to Setup KerberosIV Clients and Servers under OpenBSD

As a user/administrator of OpenBSD systems, you are fortunate that KerberosIV is a pre-installed component of the default system. Here is a guide to setting up both the Kerberos realm server, as well as a client.

An ***EXTREMELY*** important point to remember is that Kerberos clients and servers must have their system clocks synchronized. If there is more than a 5 minute time skew, you will receive weird errors that do not immediately reveal themselves to be caused by time skew, such as:

```
kinit: Can't send request (send_to_kdc)
```

Another more accurate error is:

```
kauth: Time is out of bounds (krb_rd_req)
```

An easy way to synchronize system clocks is with `ntpd`, available in the ports tree at `/usr/ports/net/ntp/`.

This FAQ entry assumes you have prior knowledge of the Kerberos concepts. For a great, easy to understand, reference, see:

- [The FreeBSD handbook](#)
- Use the command `info kth-krb`
- [Designing an Authentication System: a Dialogue in Four Scenes](#)
- [Papers and Documentation Describing KerberosIV](#)

Or the book

- [Network Security Private Communication in a Public World \[Kaufman, Perlman, Speciner, 1995\]](#)

How to setup the Kerberos IV REALM and SERVER

We will be setting up the CIARASYSTEMS.COM realm, with avalanche.ciarasystems.com as the main server.

To start off, we will need to edit our configuration files. These files are located at `/etc/kerberosIV/`. The two files we are concerned about are `krb.realms` and `krb.conf`. Let's start off with `krb.conf`.

```
# cat krb.conf
CIARASYSTEMS.COM
CIARASYSTEMS.COM avalanche.ciarasystems.com admin server
```

As you can see, this tells kerberos that the domain is CIARASYSTEMS.COM (or logical realm) and that within that domain, avalanche is the administration server. Next we will look at `krb.realms`. For more information on this refer to [krb.conf\(5\)](#).

```
# cat krb.realms
avalanche.ciarasystems.com      CIARASYSTEMS.COM
.ciarasystems.com              CIARASYSTEMS.COM
```

`krb.realms` provides a translation from a hostname to the Kerberos realm name for the services provided by that host. Each line of the translation file is in one of the following forms (domain_name should be of the form .XXX.YYY). So in this example, avalanche is the hostname of a computer on the CIARASYSTEMS.COM realm. And .ciarasystems.com is the domain name on the realm CIARASYSTEMS.COM. Again, for further information read the [krb.realms\(5\)](#) man page.

Next we will run [kdb_init\(8\)](#) to create the initial Kerberos database.

```
# kdb_init
Realm name [default NO.DEFAULT.REALM ]: CIARASYSTEMS.COM
You will be prompted for the database Master Password.
It is important that you NOT FORGET this password.

Enter Kerberos master password: not shown
Verifying password -
Enter Kerberos master password:
```

Next we need to use [kstash\(8\)](#) which is used to save the Kerberos key distribution center (KDC) database master key in the master key cache file.

```
# kstash
Enter Kerberos master password:

Current Kerberos master key version is 1.

Master key entered.  BEWARE!
Wrote master key to /etc/kerberosIV/master_key
```

This saves the encrypted master password in `/etc/kerberosIV/master_key`.

Next, we need two principals to be added to the database for each system that will be secured with Kerberos. Their names are `kpasswd` and `rcmd`. These two principals are made for each system, with the instance being the name of the individual system. These daemons, `kpasswd` and `rcmd` allow other systems to change Kerberos passwords and run commands like `rcp`, `rlogin` and `rsh`.

```
# kdb_edit
Opening database...

Enter Kerberos master key:

Current Kerberos master key version is 1.

Master key entered.  BEWARE!

Previous or default values are in [brackets] ,
enter return to leave the same, or new value.

Principal name: passwd
Instance: avalanche

<Not found>, Create [y] ? y

Principal: passwd, Instance: avalanche, kdc_key_ver: 1
New Password:          <----- Use 'RANDOM' as password
Verifying password -
New Password:

Random password [y] ? y
```

```
Principal's new key version = 1
Expiration date (enter yyyy-mm-dd) [ 1999-12-31 ] ? 2003-12-31
Max ticket lifetime (*5 minutes) [ 255 ] ?
Attributes [ 0 ] ?
Edit O.K.
```

```
Principal name: rcmd
Instance: avalanche
```

```
<Not found>, Create [y] ? y
Principal: rcmd, Instance: avalanche, kdc_key_ver: 1
New Password:          <----- Use 'RANDOM' as password
Verifying password -
New Password:
```

```
Random password [y] ? y
```

```
Principal's new key version = 1
Expiration date (enter yyyy-mm-dd) [ 1999-12-31 ] ? 2003-12-31
Max ticket lifetime (*5 minutes) [ 255 ] ?
Attributes [ 0 ] ?
Edit O.K.
Principal name:          <----- Hit <ENTER> to end
```

A srvtab file is the service key file. These must be extracted from the Kerberos key distribution center database in order for services to authenticate using Kerberos. For each hostname specified on the command line, [ext_srvtab\(8\)](#) creates the service key file hostname-new-srvtab, containing all the entries in the database with an instance field of hostname.

```
# ext_srvtab avalanche

Enter Kerberos master password:

Current Kerberos master key version is 1.

Master key entered. BEWARE!
Generating 'avalanche-new-srvtab'....

# mv avalanche-new-srvtab srvtab
# chmod 600 srvtab
```

Now we can add users to our database.

```
# kdb_edit
Opening database...

Enter Kerberos master key:

Current Kerberos master key version is 1.

Master key entered. BEWARE!
Previous or default values are in [brackets] ,
enter return to leave the same, or new value.

Principal name: jeremie
Instance:

<Not found>, Create [y] ? y

Principal: jeremie, Instance: , kdc_key_ver: 1
New Password:          <---- enter a secure password here
Verifying password

New Password:          <---- re-enter the password here
Principal's new key version = 1
Expiration date (enter yyyy-mm-dd) [ 2000-01-01 ] ?
Max ticket lifetime (*5 minutes) [ 255 ] ?
Attributes [ 0 ] ?
Edit O.K.
Principal name:          <---- null entry here will cause an exit
                          or you can add more entries.
```

So now all the Kerberos particulars are setup. All that is left is to enable boot-time loading of the Kerberos server and to enable Kerberized-daemons.

In /etc/rc.conf, set:

```
kerberos_server=YES
```

In /etc/inetd.conf, uncomment:

```
telnet      stream tcp      nowait root    /usr/libexec/telnetd  telnetd -k
```

```

klogin      stream tcp      nowait root    /usr/libexec/rlogind  rlogind -k
kshell      stream tcp      nowait root    /usr/libexec/rshd     rshd -k
kauth       stream tcp      nowait root    /usr/libexec/kauthd   kauthd

```

Then, either reboot, or:

```

# kill -HUP `cat /var/run/inetd.pid`
# /usr/libexec/kerberos >> /var/log/kerberos.log &
# /usr/libexec/kadmind -n >> /var/log/kadmind.log &

```

Note: this is a rather simple server setup. Usually, redundant servers are setup (as slave servers) so that if one server goes down, all the services that depend on Kerberos don't go down. We can also add 'su' privileges to a specific principal, see [the FreeBSD Handbook](#).

How to kerberize your client workstation

We will be setting the workstation named *gatekeeper* to be in the CIARASYSTEMS.COM realm, with *avalanche.ciarasystems.com* as the main server.

To start off, we need to setup our *krb.conf* and *krb.realms* like the above machine. This is so *gatekeeper* will know what server is the KDC and what domain it is on. Again here are the file contents.

```

# cat krb.conf
CIARASYSTEMS.COM
CIARASYSTEMS.COM avalanche.ciarasystems.com admin server

# cat krb.realms
avalanche.ciarasystems.com      CIARASYSTEMS.COM
.ciarasystems.com              CIARASYSTEMS.COM

```

Now that is set up, we need to initialize kerberos. To obtain a ticket you use [kinit\(1\)](#).

```

$ kinit
The OpenBSD Project (gatekeeper)
Kerberos Initialization
Kerberos name: jeremie
Password:

```

Now we have identified we can list our tickets with [klist\(1\)](#).

```

$ klist
Ticket file:  /tmp/tkt1000
Principal:    jeremie@CIARASYSTEMS.COM

    Issued            Expires            Principal
Jun 28 01:03:25  Jun 28 11:03:25  krbtgt.CIARASYSTEMS.COM@CIARASYSTEMS.COM

```

Looks like we are set now. All that's left to do is test it. Here we will test it with [rlogin\(1\)](#) and [telnet\(1\)](#).

```

$ telnet avalanche
Trying 192.168.0.38...
Connected to avalanche.
Escape character is '^'.
[ Trying mutual KERBEROS4 ... ]
[ Kerberos V4 accepts you ]
[ Kerberos V4 challenge successful ]
Last login: Sun Jun 27 22:52:25 on tty1 from gatekeeper
Warning: no Kerberos tickets issued.
OpenBSD 3.2 (GENERIC) #25: Thu Oct 3 19:51:53 MDT 2002

```

and

```

$ rlogin avalanche
Last login: Sun Jun 27 22:53:39 on tty1 from gatekeeper
Warning: no Kerberos tickets issued.
OpenBSD 3.2 (GENERIC) #25: Thu Oct 3 19:51:53 MDT 2002

```

We can tell that it is indeed using Kerberos to authenticate the rlogin session. To get rid of any tickets issued, you would use [kdestroy\(1\)](#). For example:

```

$ kdestroy
Tickets destroyed.
$ rlogin avalanche
krcmd: No ticket file (tf_util)
rlogin: warning, using standard rlogin: can't provide Kerberos auth data.
avalanche: Connection refused

```

Do not worry about 'Warning: no Kerberos tickets issued.' This is because we're only doing kerberos authentication, not ticket passing. If you want ticket passing, use [OpenSSH](#) which has support. Stock KerberosIV doesn't have support for tgt passing, either - only the AFS kaserver's implementation of krb4, since the regular KerberosIV kdc checks client IP address listed in the ticket.

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 account on your system of "ftp". This account shouldn't 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 [ftp\(8\)](#) 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 `echo /usr/bin/false >> /etc/shells`. Also if you wish for that shell to show up during the adduser questions, you need to modify */etc/adduser.conf*.

```
# adduser
Use option ``-silent'' if you don't want see all warnings and questions.

Reading /etc/shells
Check /etc/master.passwd
Check /etc/group

Ok, let's go.
Don't worry about mistakes. I will give you the chance later to correct any input.
Enter username [a-z0-9_]: ftp
Enter full name []: anonymous ftp
Enter shell csh false ksh nologin sh tcsh zsh [sh]: false
Uid [1002]:
Login group ftp [ftp]:
Login group is ``ftp''. Invite ftp into other groups: guest no
[no]: no
Enter password []:
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
HOME:      /home/ftp
Shell:     /usr/bin/false
OK? (y/n) [y]: y
Added user ``ftp''
Copy files from /usr/share/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 [ftp\(8\)](#) 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

With ftpd you can choose to either run it from inetd or the rc scripts can kick it off. 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/tcpd    ftpd -llUSA
```

Note - For people using HIGH traffic ftp servers, you might want to not invoke ftpd from inetd.conf. The best option is to comment the ftpd line from inetd.conf and start ftpd from rc.conf 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.

```
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.

Other relevant files

- */etc/ftpwelcome* - This holds the Welcome message for people once they have connected to your ftp server.
- */etc/motd* - This holds the message for people once they have successfully logged into your ftp server.
- *.message* - This file can be placed in any directory. It will be shown once a user enters that directory.

10.14 - Confining users to their home dir's in ftpd(8)

OpenBSD's [ftpd\(8\)](#) is setup by default to be able to handle this very easily. This is accomplished via the file */etc/ftproot*. Since users cannot always be trusted, it might be necessary to restrain them to their home directories. This behavior is NOT on by default. Here is an example of what the default behavior is like.

```

$ ftp localhost
Connected to localhost.
220 oshibana FTP server (Version 6.4/OpenBSD) ready.
Name (localhost:ericj): ericj
331 Password required for ericj.
Password: *****
230- OpenBSD 3.2 (GENERIC) #25: Thu Oct  3 19:51:53 MDT 2002
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> cd /
250 CWD command successful.
ftp> ls
227 Entering Passive Mode (127,0,0,1,60,7)
150 Opening ASCII mode data connection for 'file list'.
alroot
bin
dev
etc
home
mnt
root
sbin
stand
tmp

```

```
usr
var
bsd
sys
boot
226 Transfer complete.
ftp> quit
221 Goodbye.
```

As you can see here, access is granted to the whole server. In a perfect world this is ok, where all users can be trusted, but this isn't so. To limit a user, simply add their name to the file `/etc/ftproot`. Here is an example showing user "ericj" being restricted.

```
$ cat /etc/ftproot
#      $ OpenBSD: ftproot,v 1.3 1996/07/18 12:12:47 deraadt Exp $
#
# list of users (one per line) given ftp access to a chrooted area.
# read by ftpd(8).
ericj
```

This is enough to keep the user "ericj" from escaping from his own directory. As you can see in the next example. The `/` directory has suddenly changed to his home dir!

```
$ ftp localhost
Connected to localhost.
220 oshibana FTP server (Version 6.4/OpenBSD) ready.
Name (localhost:ericj): ericj
331 Password required for ericj.
Password: *****
230 User ericj logged in.
Remote system type is UNIX.
Using binary mode to transfer files.
ftp> cd /
250 CWD command successful.
ftp> ls
227 Entering Passive Mode (127,0,0,1,92,171)
150 Opening ASCII mode data connection for 'file list'.
.login
.mailrc
.profile
.rhosts
.ssh
.cshrc
work
mail
src
226 Transfer complete.
ftp> quit
221 Goodbye.
```

10.15 - Applying patches in OpenBSD

The OpenBSD source tree is constantly changing and improving, along with this fixes to common problems are often made and patches released to the public. These patches appear on the errata web page located at <http://www.openbsd.org/errata.html>, and are separated into categories. These categories correspond to patches that should be applied to different architectures or architecture independent patches.

Note, however, that patches aren't made for new additions to OpenBSD, and are only done for important reliability fixes or security problems that should be addressed right away, although the choice to do so is, as always, up to the administrator.

For the examples I will be patching [talkd\(8\)](#) with a security fix from the patch obtained from [errata.html](#).

How are these patches different from what I would find in the CVS tree?

All patches posted at <http://www.openbsd.org/errata.html> are patches directly against the latest release's source tree. Patches against the latest CVS tree might also include other changes that wouldn't be wanted on a release system.

Getting your system ready to be patched.

Patches for the OpenBSD Operating System are distributed as diffs, which are text files that hold differences to the original source code. They are **NOT** distributed in binary form. This means that to patch your system you must have the source code from the **RELEASE** version of OpenBSD readily available. This does not mean that you must have ALL source code to the OpenBSD operating system to patch your system, but must have all code for the program which you are patching. For instance, if you are patching the kernel you must have all source for the kernel on hand.

[cvs\(1\)](#) is a very handy tool that can be used to grab only the source that you need via any of the anonymous cvs servers located around the world. You can get a listing of these servers at <http://www.openbsd.org/anoncv.html>.

To retrieve the source code of 3.2-release using [cvs\(1\)](#), you would use the following line:

```
$ cvs -d anoncvs@anoncvs5.usa.openbsd.org:/cvs co -rOPENBSD_3_2_BASE src/libexec/talkd/
cvs server: Updating src/libexec/talkd
U src/libexec/talkd/announce.c
U src/libexec/talkd/talkd.c
U src/libexec/talkd/talkd.h
```

To find the CVS path to the code that you need, you can find this in the patch on the *Index:* line. In this case, the CVS path was *src/libexec/talkd/*. Always check out the revision of OPENBSD_version_number_BASE. Without "_BASE" you will be checking out the stable branch, which might contain other changes that will interfere. If you are already tracking the patch branch, the patches should already be in that source, however you should always check and make sure. You can always look at <http://www.openbsd.org/plus.html> to see which patches have been applied to the patch branch. If the patches haven't been applied yet, you will need to grab the latest release source using the commands above.

Also, for those users that bought official OpenBSD CDs, you can get the source code directly off of the CD. Refer to the CD insert on how to extract the source from the CD. In which case you won't need to obtain the source via anoncvs.

```
Apply by doing:
  cd /usr/src
  patch -p0 < 026_talkd.patch
  cd libexec/talkd
  make obj && make depend && make && make install

Index: libexec/talkd/announce.c <----- Path to sources
=====
RCS file: /cvs/src/libexec/talkd/announce.c,v
retrieving revision 1.8
retrieving revision 1.9
diff -u -r1.8 -r1.9
--- libexec/talkd/announce.c    1998/08/18 03:42:10    1.8
+++ libexec/talkd/announce.c    2000/07/06 00:01:45    1.9
@@ -160,6 +160,6 @@
         *(bptr++) = '\n';
     }
     *bptr = '\0';
-    fprintf(tf, big_buf);
+    fprintf(tf, "%s", big_buf);
     fflush(tf);
 }
```

Once you've obtained the proper sources, you can obtain the patch and place it in *src/*

Applying Patches

```
$ cd /usr/src
$ patch -p0</path/to/026_talkd.patch
Hmm... Looks like a unified diff to me...
The text leading up to this was:
-----
|Apply by doing:
|  cd /usr/src
|  patch -p0 < 026_talkd.patch
|  cd libexec/talkd
|  make obj && make depend && make && make install
|
|Index: libexec/talkd/announce.c
|=====
|RCS file: /cvs/src/libexec/talkd/announce.c,v
|retrieving revision 1.8
|retrieving revision 1.9
|diff -u -r1.8 -r1.9
|--- libexec/talkd/announce.c    1998/08/18 03:42:10    1.8
|+++ libexec/talkd/announce.c    2000/07/06 00:01:45    1.9
|-----
Patching file libexec/talkd/announce.c using Plan A...
Hunk #1 succeeded at 160. <----- Patch Succeeded
done
$ cd /usr/src/libexec/talkd/
$ ls
CVS          announce.c  print.c     table.c     talkd.c
Makefile     announce.c.orig process.c   talkd.8     talkd.h
$ make obj && make depend && make
making /home/ericj/lsrc/src/libexec/talkd/obj
mkdep -a /home/ericj/lsrc/src/libexec/talkd/talkd.c /home/ericj/lsrc/src/libexec/talkd/announce.c
/home/ericj/lsrc/src/libexec/talkd/process.c /home/ericj/lsrc/src/libexec/talkd/table.c
/home/ericj/lsrc/src/libexec/talkd/print.c
cc -O2      -c /home/ericj/lsrc/src/libexec/talkd/talkd.c
cc -O2      -c /home/ericj/lsrc/src/libexec/talkd/announce.c
cc -O2      -c /home/ericj/lsrc/src/libexec/talkd/process.c
cc -O2      -c /home/ericj/lsrc/src/libexec/talkd/table.c
cc -O2      -c /home/ericj/lsrc/src/libexec/talkd/print.c
```



```
cc -o ntalkd talkd.o announce.o process.o table.o print.o
nroff -Tascii -mandoc /home/ericj/lsrc/src/libexec/talkd/talkd.8 > talkd.cat8
$ sudo make install
install -c -s -o root -g bin -m 555 ntalkd /usr/libexec
install -c -o root -g bin -m 444 talkd.cat8 /usr/share/man/cat8/talkd.0
/usr/share/man/cat8/ntalkd.0 -> /usr/share/man/cat8/talkd.0
```

Once you have done that, you should restart that service.

10.16 - Tell me about this chroot() Apache?

Starting with OpenBSD 3.2, the Apache [httpd\(8\)](#) server is [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. 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 user?

Put bluntly, [chroot\(2\)](#)ing Apache is something new, and many older applications and system configurations will not work as before.

- **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 part 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/home`.
- **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 its own log files once privileges are dropped. [httpd\(8\)](#) must be stopped and restarted:

```
# apachectl stop && apachectl start
```

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 considerable programming knowledge -- most users will find it easier to just disable the [chroot\(2\)](#) feature until they are updated.

In some cases, the application or configuration can be altered to run within the [chroot](#). In other cases, you will simply have to disable this feature using the `-u` option for [httpd\(8\)](#) in [/etc/rc.conf](#)

10.17 - I don't like the standard root shell!

The default shell for `root` on OpenBSD is [csh](#), due primarily to tradition. There is no requirement that OpenBSD have [csh\(1\)](#) for the root login (though keep reading before changing it).

Some users who come from other Unix-like operating systems find [csh\(1\)](#) unfamiliar, and ask if and how they can change it. There are a few options:

- **Don't login as root!** Between [su](#) and [sudo](#), there should be few reasons for users to log in as `root` for most applications after initial setup.
- **Invoke your favorite shell after login:** If you like [ksh\(1\)](#) or any other shell, just invoke it from the default shell.
- **Change the root shell:** This can be done using [chsh](#) or [vipw](#).

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.

It is sometimes said one should never change the root shell, though there is no reason not to in OpenBSD. But again, this shouldn't be an issue -- just don't log in as root.

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 [ports](#) or [packages](#).

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`.

[\[FAQ Index\]](#) [\[To Section 9 - Migrating from Linux\]](#) [\[To Section 11 - Performance Tuning\]](#)



www@openbsd.org

\$OpenBSD: faq10.html,v 1.86 2003/04/04 17:49:08 nick Exp \$



[\[FAQ Index\]](#) [\[To Section 10 - System Administration\]](#) [\[To Section 12 - For Advanced Users\]](#)

11 - Performance Tuning

Table of Contents

- [11.1 - Networking](#)
 - [11.2 - Disk I/O](#)
 - [11.4 - Hardware Choices](#)
 - [11.5 - Why aren't we using async mounts?](#)
 - [11.6 - Tuning your monitor resolution under XFree86](#)
-

If you run a busy server, gateway or firewall, you may wish or need to adjust some of the default parameters to get optimal performance. The [options\(4\)](#) man page talks about the kernel options presented. These options are placed in the kernel configuration file before you compile a custom kernel, and some are settable via [config\(8\)](#). For more details, see [FAQ 5](#).

11.1 - Networking

A parameter you may need to change for a busy server, gateway or firewall is NMBCLUSTERS. This controls the size of the kernel mbuf cluster map. On your computer, if you get messages like "mb_map full", you need to increase the value of this parameter. If traffic on a network interface stops for no apparent reason, this may also be a sign that you need to increase the value of NMBCLUSTERS. A reasonable value on the i386 port with most 100Mbps ethernet interfaces (no matter how many the machine has) is 8192.

option NMBCLUSTERS=8192

The default number of NMBCLUSTERS varies from platform to platform, ranging from 256 to 2048. It is set in a platform-dependent header file unless overridden by an option line in a kernel config file.

Starting with OpenBSD 3.1, NMBCLUSTERS can be set using [config\(8\)](#), which eliminates the need to compile a kernel and reboot solely to change this option:

```
# config -e -o bsd.new /bsd
OpenBSD 3.2 (GENERIC) #25: Thu Oct  3 19:51:53 MDT 2002
  deraadt@i386.openbsd.org:/usr/src/sys/arch/i386/compile/GENERIC
Enter 'help' for information
ukc> nmbclust
nmbclusters = 2048
ukc> nmbclust 8192
nmbclusters = 8192
ukc> quit
```

```
Saving modified kernel.
```

This will create a new kernel, `bsd.new`, in the current directory, *and* modifies the `NMBCLUSTERS` parameter of the running kernel. If all goes well, don't forget to copy `bsd.new` to `bsd` before rebooting.

11.2 - Disk I/O

Disk I/O speed 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 several options to increase the speed of your disk operations and provide fault tolerance.

Table Of Contents

- [CCD](#) - Concatenated Disk Driver.
- [RAID](#)
- [Filesystem Buffer](#)
- [Soft Updates](#)
- [Size of the namei\(\) cache](#)

11.2.1 - CCD

The first option is the use of [ccd\(4\)](#), the Concatenated Disk Driver. This allows you to join several partitions into one virtual disk (and thus, you can make several disks look like one disk). This concept is similar to that of LVM (logical volume management), which is found in many commercial Unix flavors.

If you are running GENERIC, `ccd` is already enabled (in `/usr/src/sys/conf/GENERIC`). If you have customized your kernel, you may need to return it to your kernel configuration. Either way, a line such as this should be in your configuration file:

```
pseudo-device    ccd      4          # concatenated disk devices
```

The above example gives you up to 4 `ccd` devices (virtual disks). Now you need to figure out which partitions on your real disks you want to dedicate to `ccd`. Use `disklabel` to mark these partitions as type 'ccd'. On some architectures, `disklabel` may not allow you to do this. In this case, mark them as 'ffs'.

If you are using `ccd` to gain performance by striping, note that you will not get optimum performance unless you use the same model of disks with the same `disklabel` settings.

Edit `/etc/ccd.conf` to look something like this: (for more information on configuring `ccd`, look at [ccdconfig\(8\)](#))

```
# Configuration file for concatenated disk devices
#
# ccd  ileave  flags  component devices
ccd0  16      none  /dev/sd2e /dev/sd3e
```

To make your changes take effect, run

```
# ccdconfig -C
```

As long as `/etc/ccd.conf` exists, `ccd` will automatically configure itself upon boot. Now, you have a new disk, `ccd0`, a combination of `/dev/sd2e` and `/dev/sd3e`. Just use `disklabel` on it like you normally would to make the partition or partitions you want to use. Again, don't use the 'c' partition as an actual partition that you put stuff on. Make sure your usable partitions are at least one cylinder off from the beginning of the disk.

11.2.2 - RAID

Another solution is [raid\(4\)](#), which will have you use [raidctl\(8\)](#) to control your raid devices. OpenBSD's RAID is based upon Greg Oster's [NetBSD port](#) of the CMU [RAIDframe](#) software. OpenBSD has support for RAID levels of 0, 1, 4, and 5.

With `raid`, as with `ccd`, support must be in the KERNEL. Unlike `ccd`, support for RAID is not found in `GENERIC`, so it must be compiled into your kernel (RAID support adds some 500K to the size of an `i386` kernel).

```
pseudo-device    raid    4        # RAIDframe disk device
```

Setting up RAID on some operating systems is confusing and painful to say the least. Not so with RAIDframe. Read the [raid\(4\)](#) and [raidctl\(8\)](#) man pages to get full details. There are many options and possible configurations available, and a detailed explanation is beyond the scope of this document.

11.2.3 - Filesystem Buffer

For file servers with memory to spare, you can increase `BUFCACHEPERCENT`. That is, what percentage of your RAM should you use as a file system buffer. This option may change when the Unified Buffer Cache is completed and is part of OpenBSD. In the mean time, to increase `BUFCACHEPERCENT`, you should add a line to your kernel configuration like this:

```
option BUFCACHEPERCENT=30
```

Of course you can make it as low as 5 percent (the default) or as high as 50 percent (or more.)

Starting with OpenBSD 3.1, `BUFCACHEPERCENT` can be set using [config\(8\)](#), as [above](#), which eliminates the need to compile a kernel solely to change this option.

11.2.4 - Soft updates

Another 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. So, a system crash should not require [fsck\(8\)](#) upon boot up, but simply a background version of `fsck` that makes changes to the `metainfo` in RAM (a la `softupdates`). This means rebooting a server is much faster, as you don't have to wait for `fsck`! (OpenBSD does not have this feature yet.) You can read more about `softupdates` in the [Softupdates FAQ](#) entry.

11.2.5 - Size of the `namei()` cache

Note: previously, the [options\(4\)](#) manual page recommended to set the `NVNODE=integer` kernel option. This is no

longer recommended; you should now use the [sysctl\(8\)](#) command instead.

The name-to-inode translation (a.k.a., namei()) cache controls the speed of pathname to [inode\(5\)](#) translation. By default, this cache has $NPROC * (80 + NPROC / 8)$ entries. NPROC is set to $20 + 16 * MAXUSERS$; see the [config\(8\)](#) manual page for an explanation on the `maxusers` kernel configuration parameter. 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\(5\)](#).

11.4 - Hardware choices

(Note- this section is heavily centered around the i386, or PC, architecture. That is to say... other architectures don't give you quite as many choices!)

The performance of your applications depends heavily on your OS and the facilities it provides. This may be part of the reason that you are using OpenBSD. The performance of your applications also depends heavily on your hardware. For many folks, the Price/Performance ratio of a brand new PC with a Intel Pentium IV or AMD Athlon processor is much better than the Price/Performance ratio of a Sun UltraSparc 60! Of course, the price of OpenBSD can't be beaten.

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, when you skimp on the price of a PC, you lose in quality!

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

- **Use multiple disks.** Instead of buying one 20GB disk, buy multiple 9GB 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 will get more reliability and faster data access with RAID.
- **Use SCSI if you need very high disk IO speeds.** IDE disks normally run at 5400 RPM to 7200 RPM. Using high end IDE disks, it may be unreasonable to expect more than 15 to 20 megabytes per second of throughput from a single disk. Using high end SCSI disks (higher cost 10k RPM to 15k RPM disks), you can achieve performance higher than this. Conversely, if you are using medium or low end SCSI disks, this is a waste of money, and IDE will serve you just as well, if not better.

If you are building a server, and you need more than 20GB of disk space, you may want to consider SCSI. IDE limits you to two disks per controller. Concurrent access to these two disks may have a negative impact on the I/O performance of these disks. Wide SCSI limits you to 15 per controller, and has better support for concurrent access than IDE. While SCSI costs more, the flexibility and performance can justify these costs in some environments.

- **Use SDRAM instead of DRAM.** This option applies mainly to PCs. Most other architectures don't give you a choice of what kind of RAM you can use. Several PCs still do. You will get better performance with SDRAM versus DRAM (SIMMs). If your system supports RDRAM, DDR or some other new type of RAM, then you are even further ahead...
- **Use ECC or parity RAM.** Parity adds some functionality to see if the data in RAM has been corrupted. ECC extends this functionality and attempts to correct some bit corruption errors on the fly. This option applies mainly to PCs. Most other architectures simply require parity or ECC capable RAM. Several non-PC computers won't even boot with non-parity RAM. If you aren't using ECC/parity RAM, you may get data corruption and other abnormalities. Several manufacturers of "cheap PC RAM" don't even make an ECC variety! This will help you avoid them! PC manufacturers

often sell several product lines, divided around "servers" and "workstations." The servers will incorporate ECC RAM into their architecture. Unix workstation manufacturers have been using parity (and now ECC) for several years in all of their product lines.

- **Avoid ISA devices.** While most folks avoid ISA devices because they are generally hard to configure and out of date, there are still plenty in existence. If you are using the ISA bus for your disk or network controllers, (or even worse, for both) remember that the ISA bus itself can be a performance bottleneck. If you need speed, look into PCI. Of course, there are still several ISA bus cards that work just fine. Unfortunately, most of these are sound cards and serial port cards.
- **Avoid cheap PCI network adapters.** OpenBSD supports a plethora of cheap PCI network adapters. These adapters work great in home systems, and also 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 PCI network adapter. Unfortunately, some expensive brand adapters (such as the 3com XL series) are not much better than the cheap adapters. One favourite 10/100Mbps adapter is the Intel EtherExpress PRO/100.

11.5 - 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 is 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.

11.6 - Tuning your monitor resolution under XFree86

Getting an X server working at an acceptable resolution with many multi-sync monitors is possible. If anyone has tried to do this with the standard xf86config or XF86Setup utilities, they probably didn't get the best possible results. One of the more painful aspects is simply getting your monitor running with your preferred resolution, and then getting the vertical scan rate set to at least 72-75 Hz, a rate where the screen flicker is much less visible to humans. Conversely, what if you are trying to set the vertical scan rate so it is very low? You can set it at 50 Hz so that it can be captured on to video without flicker, but the methods to do this are non-obvious with the standard XFree86 tools and documentation.

Finally, at the resolutions many people normally use with inexpensive VGA monitors (800x600, 1024x768, 1152x900, 1280x1024), it is possible (at least on newer monitors) to use vertical scan rates of 85Hz and above, to achieve an extremely clean, palatable picture. The XFree86 X server has a mechanism which allows you to describe in detail the video mode you want to use, this is the ModeLine. A ModeLine has four sections, a single number for the pixel clock, four numbers for horizontal timings, four

numbers for vertical timings, and an optional section with a list of flags specifying other characteristics of the mode (such as Interlace, DoubleScan, and more... see the XF86Config(5) manual page for more ModeLine details).

Generating a ModeLine is a black art... Luckily, there are several scripts which can do this for you. One is [Colas XFree86 ModeLine Generator](#). Another is [The XFree86 Modeline Generator](#) hosted at SourceForge, and there are several others available on [Freshmeat](#). Before you can use these ModeLine generators, you need to figure out the vertical and horizontal sync limits for your monitor. This is often documented in the manual, or on the manufacturer's web site. If you can't find either of those, simply search the web for the monitor make and model, several people have been kind enough to compile lists with this information.

For example, say you have a Dell D1226H monitor. You searched in agony at Dell's web site to find that it has a 30-95 kHz horizontal scan range, and a 50-160 Hz vertical scan range. Visit the ModeLine generator page, enter this information. Next, you need to enter the minimum vertical scan rate you want. Any rate at or above 72 Hz should generally have low visible flicker. As you go higher, the clearer and crisper your screen image becomes.

With all of these bits of information, the script will generate a ModeLine for every possible 4x3 resolution which your monitor can support, above the minimum vertical scan rate which you enter. If someone enters the Dell specs above and a 75 Hz vertical scan minimum, the script gives out something like the following:

```
ModeLine "320x240" 20.07 320 336 416 448 240 242 254 280 #160Hz
ModeLine "328x246" 20.86 328 344 424 456 246 248 260 286 #160Hz
...
ModeLine "816x612" 107.39 816 856 1056 1136 612 614 626 652 #145Hz
ModeLine "824x618" 108.39 824 864 1064 1144 618 620 632 658 #144Hz
ModeLine "832x624" 109.38 832 872 1072 1152 624 626 638 664 #143Hz
...
ModeLine "840x630" 109.58 840 880 1080 1160 630 632 644 670 #141Hz
ModeLine "848x636" 110.54 848 888 1088 1168 636 638 650 676 #140Hz
...
ModeLine "1048x786" 136.02 1048 1096 1336 1432 786 788 800 826 #115Hz
ModeLine "1056x792" 136.58 1056 1104 1344 1440 792 794 806 832 #114Hz
ModeLine "1064x798" 137.11 1064 1112 1352 1448 798 800 812 838 #113Hz
...
ModeLine "1432x1074" 184.07 1432 1496 1816 1944 1074 1076 1088 1114 #85Hz
ModeLine "1576x1182" 199.86 1576 1648 2008 2152 1182 1184 1196 1222 #76Hz
ModeLine "1584x1188" 198.93 1584 1656 2016 2160 1188 1190 1202 1228 #75Hz
```

Now, this monitor claims to do 1600x1200 @ 75 Hz, but the script does not say this is within 75 Hz. So, if you really want exactly 1600x1200, go down a notch with your minimum vertical rate... (Here, we go down to 70 Hz)

```
ModeLine "1592x1194" 197.97 1592 1664 2024 2168 1194 1196 1208 1234 #74Hz
ModeLine "1600x1200" 199.67 1600 1672 2032 2176 1200 1202 1214 1240 #74Hz
ModeLine "1608x1206" 198.65 1608 1680 2040 2184 1206 1208 1220 1246 #73Hz
ModeLine "1616x1212" 197.59 1616 1688 2048 2192 1212 1214 1226 1252 #72Hz
ModeLine "1624x1218" 199.26 1624 1696 2056 2200 1218 1220 1232 1258 #72Hz
ModeLine "1632x1224" 198.15 1632 1704 2064 2208 1224 1226 1238 1264 #71Hz
ModeLine "1640x1230" 199.81 1640 1712 2072 2216 1230 1232 1244 1270 #71Hz
ModeLine "1648x1236" 198.64 1648 1720 2080 2224 1236 1238 1250 1276 #70Hz
```

Here, we see the monitor really does 1600x1200 @ 74 Hz when the dot clock (bandwidth) is limited to 200MHz. Set the bandwidth according to the limits defined by the monitor.

Once you have your ModeLines, put them into your `/etc/XF86Config` file. Comment out the old ModeLines, so that you can use them again if the new ones don't work. Next, choose what resolution you actually want to run at. First, figure out if X is running in accelerated mode (which it does with most video cards), so you know which "Screen" section of the XF86Config to modify. Or, just modify all of the Screen sections.


```

Section "Screen"
    Driver      "Accel"
    Device      "Primary Card"
    Monitor     "Primary Monitor"
    DefaultColorDepth 32
    SubSection "Display"
        Depth    32
        Modes    "1280x1024" "1024x768"
    EndSubSection

```

The first resolution you see after the "Modes" keyword is the resolution that X is going to start in. By pressing CTRL-ALT-KEYPAD MINUS, or CTRL-ALT-KEYPAD PLUS, you can switch between any resolutions that you list here. According to the section above, X will try to start in 32-bit color mode (via the DefaultColorDepth directive, without it X will start in 8-bit color mode.) The first resolution it will try to use is 1280x1024 (it follows the order of the Modes line.) Note that "1280x1024" is just a label for the values in the ModeLine.

Note that the ModeLine generator script has options to relax its timings for older or smaller monitors, and also has the ability to provide ModeLines for specific resolutions. Depending on the type of hardware you have, it may not be very easy to use with the default options. If the picture is too tall, too wide, or too small, or is shifted horizontally or vertically, and the controls of the monitor aren't enough to correct its appearance, once can use `xvidtune(1)` to adjust the ModeLine to better fit with the monitor's timings.

On most modern monitors, there is no fixed limit on the bandwidth, thus they are often not listed anymore in the specs. What happens is that the more you go up in bandwidth, the fuzzier the screen image becomes. So you may want to put in the bandwidth of your card (also named "dotclock") to test (you cannot damage your monitor this way), and go progressively down in BW down to have a nice crisp image.

If this seems needlessly complex, that's because it is. XFree86 4.0 addresses this, and makes this process much easier since it has several built-in modes and is capable of reading back capabilities from "plug and play" monitors through DDC and DDC2.

You can download the Colas XFree86 ModeLine Generator script at: <http://koala.ilog.fr/ftp/pub/Klone/>. You need to grab the Klone interpreter, and compile it. It is in the ports as `lang/klone`. The scripts exist under the scripts directory in the Klone distribution. (The port installs them to `/usr/local/lib/klone/scripts`.)

There are two versions of the script included, the first is a CGI version identical to the web page above. The second is a non-CGI version which will take your complete XF86Config file, decode the monitor specs that you entered into `xf86config/XF86Setup` (Now, think, did you actually enter the specs for your monitor or just choose generic ones?), and fix the existing ModeLines accordingly.

[\[FAQ Index\]](#) [\[To Section 10 - System Administration\]](#) [\[To Section 12 - For Advanced Users\]](#)



www@openbsd.org

\$OpenBSD: faq11.html,v 1.39 2003/04/04 17:49:08 nick Exp \$



[\[FAQ Index\]](#) [\[To Section 11 - Performance Tuning\]](#) [\[To Section 13 - IPsec\]](#)

12 - For Advanced Users

Table of Contents

- [12.1 - Forcing DMA access for IDE disks](#)
 - [12.2 - Upgrading from various versions of OpenBSD via CVS.](#)
-

By *advanced users*, we mean to imply people who have a working knowledge of how to use and administrate a Unix-like operating system. If you follow instructions in this section without understanding what you are doing, you may cause problems in the operation of your system.

12.1 - Forcing DMA access for IDE disks

With the PCI IDE code, your chipset may not be known. If so, you will get a message like:

```
pciide0: DMA, (unused)
```

If you get this message, you can try and force DMA mode by using 'flags 0x0001' on your pciide entry in your kernel config file. That would look something like this:

```
pciide* at pci ? dev ? function ? flags 0x0001
```

After doing this, the pciide code will try to use DMA mode regardless of whether or not it actually knows how to do so with your chipset. If this works, and your system makes it through fsck and startup, it is likely that this will work for good. If this does not work, and the system hangs or panics after booting, then you can't use DMA mode (yet, until support is added for your chipset). Note that if you find the documentation for your PCI-IDE controller's chipset, this is a good start to fully supporting your chipset within the PCI-IDE code. You can look on the manufacturer's website or call them. If your PCI-IDE controller is part of your motherboard, figure out who manufactures the chipset and pursue their resources!

Note that you will know that DMA support has been enabled if you see this message:

```
cd0(pciide0:1:0): using PIO mode 3, DMA mode 1
```

This means that pciide0, channel 1, drive 0 (which appears to be an ATAPI CD-ROM) is using DMA data transfers.

[\[FAQ Index\]](#) [\[To Section 11 - Performance Tuning\]](#) [\[To Section 13 - IPsec\]](#)



www@openbsd.org

\$OpenBSD: faq12.html,v 1.34 2003/03/18 03:41:04 nick Exp \$



[\[FAQ Index\]](#) [\[To Section 12 - For Advanced Users\]](#) [\[To Section 14 - Using disks in OpenBSD\]](#)

13 - Using IPsec (IP Security Protocol)

Table of Contents

- [13.1 - What is IPsec?](#)
- [13.2 - That's nice, but why do I want to use IPsec?](#)
- [13.3 - What are the protocols behind IPsec?](#)
- [13.4 - On the wire format](#)
- [13.5 - Configuring IPsec](#)
- [13.6 - How do I set up IPsec with manual keying?](#)
- [13.7 - How do I set up isakmpd?](#)
- [13.8 - How do I use isakmpd with X.509 certificates?](#)
- [13.9 - What IKE clients are compatible with isakmpd?](#)
- [13.10 - Troubleshooting IPsec/VPN](#)
- [13.11 - Related Documentation](#)

Portions of this document were taken from:

- [ipsec\(4\)](#)
- [vpn\(8\)](#)
- [IPsec for Dummies](#) by Julian Elischer
- [ISAKMP Howto](#) by Patrick Ethier
- [X.509v3 certificates with isakmpd](#) by Jörgen Granstam

13.1 - What is IPsec?

IPsec is a set of extensions to the IP protocol family. It provides cryptographic security services. These services allow for authentication, integrity, access control, and confidentiality. IPsec provides similar services as SSL, but at the network layer, in a way that is completely transparent to your applications, and much more powerful. We say this because your applications do not have to have any knowledge of IPsec to be able to use it. You can use [any IP protocol](#) over IPsec. You can create encrypted tunnels (VPNs), or just do encryption between computers. Since you have so many options, IPsec is rather complex (much more so than SSL!)

Before you start using IPsec, we strongly recommend that you check out the "recommended reading" of [part 6](#) of the FAQ. In particular, if you don't already understand it, the [Understanding IP Addressing](#) (or [here](#)) document is highly recommended.

In a logical sense, IPsec works in any of these three ways:

- Host-to-Host
- Host-to-Network

- Network-to-Network

In every scenario that involves a network, we mean to imply router. As in, Host-to-Router (and this router controls and encrypts traffic for a particular *Network*).

As you can see, IPsec can be used to tunnel traffic for VPN connections. However, its utility reaches beyond VPNs. With a central Internet Key Exchange registry, every machine on the Internet could talk to another one and employ powerful encryption and authentication!

13.2 - That's nice, but why do I want to use IPsec?

The internet protocol, IP, aka IPv4, does not inherently provide any protection to your transferred data. It does not even guarantee that the sender is who he says he is. IPsec tries to remedy this. These services are considered distinct, but the IPsec supports them in a uniform manner.

Confidentiality

Ensure it is hard for anyone but the receiver to understand what data has been communicated. For example, ensuring the secrecy of passwords when logging into a remote machine over the Internet.

Integrity

Guarantee that the data does not get changed on the way. If you are on a line carrying invoicing data you probably want to know that the amounts and account numbers are correct and not altered while in-transit.

Authenticity

Sign your data so that others can see that it is really you that sent it. It is clearly nice to know that documents are not forged.

Replay protection

We need ways to ensure a datagram is processed only once, regardless of how many times it is received. I.e. it should not be possible for an attacker to record a transaction (such as a bank account withdrawal), and then by replaying it verbatim cause the peer to think a new message (withdrawal request) had been received. *WARNING: as per the standards specification, replay protection is not performed when using manual-keyed IPsec (e.g., when using [ipsecadm\(8\)](#)).*

13.3 - What are the protocols behind IPsec?

IPsec provides confidentiality, integrity, authenticity, and replay protection through two new protocols. These protocols are called Authentication Header (AH), and Encapsulating Security Payload (ESP).

AH provides authentication, integrity, and replay protection (but not confidentiality). The main difference between the authentication features of AH and ESP is that AH also authenticates portions of the IP header of the packet (such as the source/destination addresses). ESP authenticates only the packet payload.

ESP can provide authentication, integrity, replay protection, and confidentiality of the data (it secures everything in the packet that follows the header). Replay protection requires authentication and integrity (these two go always together). Confidentiality (encryption) can be used with or without authentication/integrity. Similarly, one could use authentication/integrity with or without confidentiality.

In practice, it is recommended that ESP be used for most applications.

13.4 - On the wire format

The **Authentication Header** (AH) comes after the basic IP header and contains cryptographic hashes of the data and identification information. The hashes can also cover the invariant parts of the IP header itself. There are several different RFCs giving a choice of actual algorithms to use in the AH, however they all must follow the guidelines specified in [RFC2402](#).

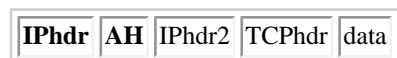
The **Encapsulating Security Payload** (ESP) header, allows for rewriting of the payload in encrypted form. The ESP header does not consider the fields of the IP header before it and therefore makes no guarantees about anything except the payload. The various types of ESP applicable must follow [RFC2406](#). An ESP header can also provide authentication for the payload, (but not the outer header).

An orthogonal (mostly) division of IPsec functionality is applied depending on whether the endpoint doing the IPsec encapsulation is the original source of the data or a gateway:

- **Transport** mode is used by a host that is generating the packets. In transport mode, the security headers are added before the transport layer (e.g. TCP, UDP) headers, before the IP header is prepended to the packet. In other words an AH added to the packet will cover the hashing of the TCP header and some fields of the end-to-end IP header, and an ESP header will cover the encryption of the TCP header and the data, but not the end-to-end IP header.
- **Tunnel** mode is used when the end-to-end IP header is already attached to the packet, and one of the ends of the secure connection is only a gateway. In this mode, the AH and ESP headers are used to cover the entire packet including the end-to-end header, and a new IP header is prepended to the packet that covers just the hop to the other end of the secure connection (though that may of course be several IP hops away).

IPsec secured links are defined in terms of **Security Associations** (SAs). Each SA is defined for a single unidirectional flow of data, and usually (ignoring multicast) from one single point to another, covering traffic distinguishable by some **unique selector**. All traffic flowing over a single SA is treated the same. Some traffic may be subject to several SAs, each of which applies some transform. Groups of SAs are called an SA **Bundle**. Incoming packets can be assigned to a particular SA by the three defining fields, (**Destination IP address, Security Parameter Index, security protocol**). SPI can be considered a cookie that is handed out by the receiver of the SA when the parameters of the connection are negotiated. The security protocol must be either AH or ESP. Since the IP address of the receiver is part of the triple, this is a guaranteed unique value. They can be found from the outer IP header and the first security header (which contains the SPI and the security protocol).

An example of a tunnel mode AH packet is:



An example of a transport mode AH packet is:



Because an ESP header cannot authenticate the outer IP header, it is useful to combine an AH and an ESP header to get the following:



This is called **Transport Adjacency**. The tunneling version would look like:



However it is not specifically mentioned in the RFC. As with Transport adjacency, this would authenticate the entire packet except a few headers in the IP header and also encrypt the payload (seen in italics). When an AH and an ESP header are directly applied together like this, the order of the headers should be as shown. It is possible in tunnel mode, to do arbitrary recursive encapsulation so that order is not specified.

13.5 - Configuring IPsec

How the IPsec systems and gateways are configured is to some extent left to the designer, however the RFC has some strong recommendations as to how this should be implemented, so as to minimize confusion.

There are two administrative entities that control what happens to a packet. One is the **Security Association Database** (SAD, referred to as TDB or TDB table throughout OpenBSD's IPsec source code) and the other is the **Security Policy Database** (SPD).

They are similar in that given a number of selectors that describe some traffic, they will deliver an entry that describes the processing needed. However, the SPD is two steps removed from the actual processing: the SPD is used for outgoing packets, to decide what SAD entries should be used, and the SAD entries in turn describe the actual process and the parameters for it. The SPD entries specify the existing SAD entries to use (if it's a bundle there can be more than 1), but if there is not already a suitable one, it is used to create new ones. The fields of the SA being created can be taken either from the SPD entry or from the packet that initiated the creation.

Outgoing packets go from the SPD entry to the specific SA, to get encoding parameters. Incoming packets get to the correct SA directly using the SPI/DestIP/Proto triple, and from there get to the SPD entry.

The SPD can also specify what traffic should bypass IPsec and what should be dropped, so it must also be consulted for incoming non-IPsec traffic. SPD entries must be explicitly ordered as several might match a particular packet, and the processing must be reproducible.

The SPD can be thought of as similar to a packet filter where the actions decided upon are the activation of SA processes. Selectors can include src and dest address, port numbers if relevant, application and user IDs if available (only on host based transport SAs), hostnames, security sensitivity levels, protocols, etc.

A SAD entry would include:

- Dest IP address
- IPsec proto (AH or ESP)
- SPI (cookie)
- Sequence counter
- Seq O/F flag
- Anti-replay window info
- AH type and info
- ESP type and info
- Lifetime info
- Tunnel/transport mode flags
- Path MTU info

A SPD entry would contain:

- Pointer to active SAs
- Selector fields

Each SA can define one ESP header and one AH header. An IPsec session must have one or the other or both, but cannot be defined with neither - otherwise there would be no headers to specify the SPI to look up the SA. The RFC doesn't say what would happen if the AH and ESP headers disagree about the SPI value. One would presume this would imply multiple SAs in a bundle.

The SPD in OpenBSD is managed through the `ipsecadm flow` command. (You would only make changes to it if you were using

manual keying.) SAD entries can be set manually with [ipsecadm\(8\)](#), however the IETF has also defined automatic mechanisms for initialization of sessions and such things as key exchange. OpenBSD implements ISAKMP automatic key exchange ([RFC2407](#), [RFC2408](#), and [RFC2409](#)) in the [isakmpd\(8\)](#) daemons.

13.6 - How do I set up IPsec with manual keying?

Manual keying is the easiest way to get started with IPsec. You can set up encryption between networks, to create VPNs using this method. After you have read this section, you may want to investigate using [/usr/share/ipsec/rc.vpn](#) to set this up for you automatically.

First, you need to ensure that the IP ESP protocol is enabled in the OpenBSD kernel. Though it is enabled by default, you should verify this fact by doing the following:

```
# sysctl net.inet.esp.enable
net.inet.esp.enable = 1
```

Similarly, if AH is required, verify that it has been enabled:

```
# sysctl net.inet.ah.enable
net.inet.ah.enable = 1
```

If necessary, these protocols may be enabled using the `-w` option to [sysctl\(8\)](#).

You can edit `/etc/sysctl.conf` to turn these on (or off) at boot time. You need to remove the `#` mark from in front of `net.inet.esp.enable` and/or `net.inet.ah.enable` (depending on which you plan to use) and make sure they are set to 1.

Note that in most cases, such as with the `rc.vpn` script or with the example below, only ESP is required. In these cases, *you do not need to enable AH*. In fact, there may be security concerns related to enabling AH if you are not using it.

Next, you will need to generate your manual keys. Since the security of the VPN is based on these keys being unguessable, it is very important that the keys be chosen using a strong random source. One practical method of generating them is by using the [random\(4\)](#) device. To produce 160 bits (20 bytes) of randomness, for example, do:

```
# openssl rand 20 | hexdump -e '20/1 "%02x"'
```

The number of bits produced is important. Different cipher types may require different sized keys.

Cipher	Key Length
DES	56 bits
3DES	168 bits
BLF	Variable (160 bits recommended)
CAST	Variable (40-128 bits recommended)
SKIPJACK	80 bits

Now, you need to set up SAs, or Security Associations. A Security Association is a combination of your IP addresses, an SPI, and your security protocol (AH and/or ESP). The IP addresses are both your own and that of your destination. The SPI, or Security Parameter Index, is a number that OpenBSD uses to classify different SAs.

These examples only use ESP to encrypt your traffic. ESP includes authentication of the contained encrypted data, but does not authenticate the surrounding IP header, as AH would. This "limited authentication" is nevertheless quite sufficient in most cases, especially for ESP in a tunnel environment.


```
# ipsecadm new esp -spi SPI_OUT -src MY_EXTERNAL_IP -dst PEER_EXTERNAL_IP -
forcetunnel -enc blf -auth sha1 -key ENC_KEY -authkey AUTH_KEY
```

Let's put this into practice with two routers, 192.168.5.1 and 192.168.25.9.

On Host 192.168.5.1:

```
# ipsecadm new esp -spi 1000 -src 192.168.5.1 -dst 192.168.25.9 -forcetunnel -
enc blf -auth sha1 -key 7762d8707255d974168cbb1d274f8bed4cbd3364 -authkey
6a20367e21c66e5a40739db293cf2ef2a4e6659f
# ipsecadm new esp -spi 1001 -dst 192.168.5.1 -src 192.168.25.9 -forcetunnel -
enc blf -auth sha1 -key 7762d8707255d974168cbb1d274f8bed4cbd3364 -authkey
6a20367e21c66e5a40739db293cf2ef2a4e6659f
```

On Host 192.168.25.9:

```
# ipsecadm new esp -spi 1001 -src 192.168.25.9 -dst 192.168.5.1 -forcetunnel -
enc blf -auth sha1 -key 7762d8707255d974168cbb1d274f8bed4cbd3364 -authkey
6a20367e21c66e5a40739db293cf2ef2a4e6659f
# ipsecadm new esp -spi 1000 -dst 192.168.25.9 -src 192.168.5.1 -forcetunnel -
enc blf -auth sha1 -key 7762d8707255d974168cbb1d274f8bed4cbd3364 -authkey
6a20367e21c66e5a40739db293cf2ef2a4e6659f
```

Notice that the SPIs are different. See [On the wire format](#) for a complete description of what the SPI is and where it is used.

Now that you have your Security Associations in place, set up your flows.

On 192.168.5.1:

So, right here, **two** flows will be created, one the local source address, which covers all packets originating from the local host to the destination, as well as a flow from the destination back to the local host.

```
# ipsecadm flow -proto esp -dst 192.168.25.9 -spi 1000 -addr 192.168.5.1
255.255.255.255 192.168.25.9 255.255.255.255
```

On 192.168.25.9:

```
# ipsecadm flow -proto esp -dst 192.168.5.1 -spi 1001 -addr 192.168.25.9
255.255.255.255 192.168.5.1 255.255.255.255
```

If you want less overhead on your Host-to-Host VPNs, creating the SPI without `-forcetunnel` will let you use transport mode (whereas, `-forcetunnel` makes sure all of the IP packet, including the IP header, are encapsulated by SPI). If either the source or destination is a network, you will have to use tunnel mode. Creating an SA to and/or from a network will automatically ensure tunnel mode SPIs are being created.

This is a simple way to start using IPsec.

You can use IPsec to tunnel private IP address spaces over the Internet. Here is a good example... We want to tunnel 192.168.99.0/24, which is behind 208.1.1.1, to 208.1.2.0/24 and 208.1.5.0/24 which are behind 208.2.2.2. These examples were generated using the [rc.vpn](#) script.

As you can see, when you are using manual keying with IPsec, you have to specify **exactly** what you want done. It won't guess for

you. Look at these examples...

On 208.1.1.1:

First, set up the security associations (SAs):

(This sets up the SPIs, encryption methods, and keys.)

```
# ipsecadm new esp -src 208.1.1.1 -dst 208.2.2.2 -forcetunnel -spi 1001 -enc
blf -auth sha1 -key 7762d8707255d974168cbb1d274f8bed4cbd3364 -authkey
67e21c66e5a40739db293cf2ef2a4e6659f
# ipsecadm new esp -src 208.2.2.2 -dst 208.1.1.1 -forcetunnel -spi 1000 -enc
blf -auth sha1 -key 7762d8707255d974168cbb1d274f8bed4cbd3364 -authkey
67e21c66e5a40739db293cf2ef2a4e6659f
```

Next, set up a flow from 208.1.1.1 to 208.2.2.2

```
# ipsecadm flow -proto esp -dst 208.2.2.2 -spi 1001 -addr 208.1.1.1
255.255.255.255 208.2.2.2 255.255.255.255
```

Next, set up a flow from 208.1.2.0/24, which is behind 208.2.2.2, to 192.168.99.0/24

```
# ipsecadm flow -proto esp -dst 208.2.2.2 -spi 1001 -addr 192.168.99.0
255.255.255.0 208.1.2.0 255.255.255.0
```

Next, set up a flow from 208.1.5.0/24, which is behind 208.2.2.2, to 192.168.99.0/24

```
# ipsecadm flow -proto esp -dst 208.2.2.2 -spi 1001 -addr 192.168.99.0
255.255.255.0 208.1.5.0 255.255.255.0
```

Now, set up a flow from 208.1.2.0/24, which is behind 208.2.2.2 to the router 208.1.1.1.

```
# ipsecadm flow -proto esp -dst 208.2.2.2 -spi 1001 -addr 208.1.1.1
255.255.255.255 208.1.2.0 255.255.255.0
```

OK, set up a flow from 208.1.5.0/24, which is behind 208.2.2.2, to the router 208.1.1.1

```
# ipsecadm flow -proto esp -dst 208.2.2.2 -spi 1001 -addr 208.1.1.1
255.255.255.255 208.1.5.0 255.255.255.0
```

Finally, set up a flow from the router 208.2.2.2 to 192.168.99.0/24

```
# ipsecadm flow -proto esp -dst 208.2.2.2 -spi 1001 -addr 192.168.99.0
255.255.255.0 208.2.2.2 255.255.255.255
```

On 208.2.2.2:

Same as before, we set up the SAs...

```
# ipsecadm new esp -src 208.2.2.2 -dst 208.1.1.1 -forcetunnel -spi 1000 -enc
blf -auth sha1 -key 7762d8707255d974168cbb1d274f8bed4cbd3364 -authkey
```

```
67e21c66e5a40739db293cf2ef2a4e6659f
# ipsecadm new esp -src 208.1.1.1 -dst 208.2.2.2 -forcetunnel -spi 1001 -enc
blf -auth sha1 -key 7762d8707255d974168cbb1d274f8bed4cbd3364 -authkey
67e21c66e5a40739db293cf2ef2a4e6659f
```

Now, this is the reverse side... Set up a flow from the router 208.2.2.2 to 208.1.1.1

```
# ipsecadm flow -proto esp -dst 208.1.1.1 -spi 1000 -addr 208.2.2.2
255.255.255.255 208.1.1.1 255.255.255.255
```

Set up a flow from the network 192.168.99.0/24, which is behind 208.1.1.1, to 208.1.2.0/24

```
# ipsecadm flow -proto esp -dst 208.1.1.1 -spi 1000 -addr 208.1.2.0
255.255.255.0 192.168.99.0 255.255.255.0
```

Set up a flow from the network 192.168.99.0/24, which is behind 208.1.1.1, to 208.1.5.0/24

```
# ipsecadm flow -proto esp -dst 208.1.1.1 -spi 1000 -addr 208.1.5.0
255.255.255.0 192.168.99.0 255.255.255.0
```

Now, set up a flow from 192.169.99.0/24, which is behind 208.1.1.1, to the router 208.2.2.2

```
# ipsecadm flow -proto esp -dst 208.1.1.1 -spi 1000 -addr 208.2.2.2
255.255.255.255 192.168.99.0 255.255.255.0
```

We're almost done... Two flows left to get 208.1.2.0/24 and 208.1.5.0/24 from the router 208.2.2.2 to the router 208.1.1.1.

```
# ipsecadm flow -proto esp -dst 208.1.1.1 -spi 1000 -addr 208.1.2.0
255.255.255.0 208.2.2.2 255.255.255.255 -ingress
# ipsecadm flow -proto esp -dst 208.1.1.1 -spi 1000 -addr 208.1.5.0
255.255.255.0 208.2.2.2 255.255.255.255 -ingress
```

If you have been using ipsecadm, and you want to get rid of any work that you've done, and start from scratch, do

```
# ipsecadm flush
```

This will flush all IPsec info (SPIs, flows, routing entries) from your system.

13.7 - How do I set up isakmpd?

If you are thinking about VPNs or other traditional applications of IPsec, you probably are going to use ISAKMP. Some commercial implementations of IPsec do not provide any manual keying ability, instead they require you to use some form of ISAKMP.

13.7.1 - What is isakmpd?

ISAKMP is the Internet Security Association and Key Management Protocol. It is sometimes referred to as IKE, or Internet Key Exchange. It is the standard key exchange mechanism for IPsec. It addresses security concerns using the methods mentioned in [RFC 2407](#), [RFC 2408](#), and [RFC 2409](#). ISAKMP manages the exchange of cryptographic keys that you would normally have to manage with [ipsecadm\(8\)](#). It employs a two-phase process for establishing the IPsec parameters between two IPsec nodes.

Phase 1 - The two ISAKMP peers establish a secure, authenticated channel upon which to communicate between two daemons. This establishes a Security Association (SA) between both hosts. **Main Mode** and **Aggressive Mode** are the methods used to establish this channel. Main Mode sends the various authentication information in a certain sequence, providing identity protection. Aggressive Mode does not provide identity protection because all of the authentication information is sent at the same time. Aggressive mode should only be used in such cases where network bandwidth is of concern, as it can expose identity information to an eavesdropper.

Phase 2 - Security Associations are negotiated on behalf of IPsec. Phase 2 establishes tunnels or endpoint SAs between IPsec hosts. **Quick Mode** is used in Phase 2 because there is no need to repeat a full authentication; Phase 1 has already established the SAs.

In brief, Phase 1 is used to get a secure channel in which to do the (quicker) phase 2 setups. There can be multiple phase 2 setups within the same phase 1 channel. Phase 2 is used to set up the actual tunnels. In Phase 1, your IPsec nodes establish a connection where they exchange authentication (Either a X.509 certificate or a pre-shared secret). This allows each end to make sure the other end is authenticated. Phase 2 is an exchange of keys to determine how the data between the two will be encrypted.

13.7.2 - How do I get started with isakmpd?

By default, OpenBSD comes with the all the necessary binaries for ISAKMP and the IPsec stack. Example configuration files may be found in `/usr/share/ipsec/isakmpd`. For the purposes of this example, copy *policy* to `/etc/isakmpd/isakmpd.policy`. Copy *VPN-east.conf* to `/etc/isakmpd/isakmpd.conf`. Here we attempt to show you how to set up a VPN (tunnel). If you want to use isakmpd between single hosts, there are other configuration files in the *samples* directory. The manual pages have detailed information.. Don't forget [isakmpd.conf\(5\)](#) and [isakmpd.policy\(5\)](#).

Although both the ESP and AH protocols are enabled in the kernel by default, you should verify the protocols you need are available by the procedure outlined in the FAQ section on [Manual Keying](#). Next, you need to edit `/etc/isakmpd/isakmpd.policy`. This file tells ISAKMP who can access IPsec. In this scenario, the policy file states that anybody who sends data using Encapsulating Security Payload (ESP), and has authenticated with the passphrase *mekmitasdigoat* (or whatever passphrase you determine), is allowed to communicate with isakmpd. You can modify this file to let ISAKMP know that we only want to allow data signed with certain digital certificates or using a certain encryption transform. You could also allow anybody to access IPsec. This is only recommended for testing. To do this, edit your policy file to contain only the following lines:

```
KeyNote-Version: 2
Authorizer: "POLICY"
```

The same policy file contains two lines that start with the \$ character. You need to remove these lines before using it; they are only for cvs.

A more useful policy file for this example looks like this:

```
KeyNote-Version: 2
Comment: This policy accepts ESP SAs from a remote that uses the right password
Authorizer: "POLICY"
Licensees: "passphrase:mekmitasdigoat"
Conditions: app_domain == "IPsec policy" &&
            esp_present == "yes" -> "true";
```

Implementing this will give you a basic VPN (tunnel) using ESP only. On host A, edit `/etc/isakmpd/isakmpd.conf`. The 249.2.2.2 sample IP address should be replaced with the external IP address of host A.

```
[General]
Retransmits=          5
Exchange-max-time=   120
Listen-on=            249.2.2.2
```

Do similar for isakmpd.conf on host B. 249.3.3.3 represents the external IP address for host B.

```
[General]
Retransmits=          5
Exchange-max-time=    120
Listen-on=            249.3.3.3
```

This is where you can set up the variables that will affect the main behavior of `isakmpd`. It is okay to use the defaults here. The **Listen-on=** value specifies the IP that `isakmpd` should listen on. Only the Internet IP of your gateway is necessary. If you have multiple external interfaces on your gateway, you could list which interfaces you want to listening on by entering them using a comma separated list.

Next, on host A, edit `isakmpd.conf` again.

```
[Phase 1]
249.3.3.3=            HostB
```

On host B:

```
[Phase 1]
249.2.2.2=            HostA
```

This section describes the IP addresses to accept in order to negotiate the phase 1 connection. Its value points to the section below (Remember that phase 1 simply authenticates the remote peer to make sure they are who they say they are). You can list multiple peers with additional lines in the format of `IP_Address= <PEER-NAME>`.

Next, on host A:

```
[Phase 2]
Connections=          HostA-HostB
```

On host B:

```
[Phase 2]
Connections=          HostB-HostA
```

This describes Phase 2 of the connection. This is the phase that determines what protocols the two peers will use to communicate.

The **Connections=** tag refers to the section below. It initiates the requirements or the accepted methods to set up Phase 2. This also tells ISAKMPD which connections to initiate once started. Note that you can have multiple sections as illustrated below if you are to connect with multiple peer hosts.

If you do not have the IP address of the remote host, you can specify a `Default=` that points to a section describing a generic entry that will be referenced by any incoming IP that is not listed in the **Connections=** tag.

On host A:

```
[HostB]
Phase=                1
Transport=            udp
Local-address=        249.2.2.2
Address=              249.3.3.3
Configuration=        Default-main-mode
Authentication=       mekmitasdigoat
```

```
#Flags=
```

On host B:

```
[HostA]
Phase=          1
Transport=      udp
Local-address=  249.3.3.3
Address=        249.2.2.2
Configuration=  Default-main-mode
Authentication= mekmitasdigoat
#Flags=
```

These represent the sections referred to by the Phase 1 section above. They each describe the requirements that the peer gateway must fulfill in order to proceed to Phase 2. There are many other options here but the ones mentioned above are the minimum requirements.

- **Phase=1** is required because the ISAKMPD code uses the same procedures to process Phase 1 and Phase 2. It must be 1 or nothing will work.
- **Transport=** gives you different possibilities for different peers. It's suggested that udp be used here so we'll leave it at that. Please note that some peers may be behind a firewall that doesn't let UDP traffic through. Obviously, this needs to be determined before setup.
- **Local-address** is the destination address that the incoming packets point to. In some cases, you can be listening on different Interfaces for Phase 1 connections. In this example, there is only 1 interface listening, therefore this is the IP of the listening interface on this peer.
- **Address=** is the address that points to the source IP of the incoming packets. This usually points to the peer gateway. This needs further explanation, because the source IP address of the peer may be unknown!
- **Configuration=** points to the section below. You can specify multiple sections like this. We use the default one specified by the sample file.
- **Authentication=** is the pre-shared secret to be used for this particular peer. It is more or less a passphrase that each peer uses. This passphrase gets passed to policy to verify whether this peer is allowed to use IPSEC with this host. If you change this phrase, you must also change it in the policy file because the sample file provides for this passphrase. If you decided to go with a minimum policy file then you can specify whatever you want here.
- **Flags=** is not currently being used. The RFCs leave room for extra options to be specified for phase 1.

There are other tags here that will allow for other options to be set. Refer to [isakmpd.conf\(5\)](#) for descriptions.

On Host A:

```
[HostA-HostB]
Phase=          2
ISAKMP-peer=    HostB
Configuration=  Default-quick-mode
Local-ID=       Net-A
Remote-ID=      Net-B
```

On Host B:

```
[HostB-HostA]
Phase=          2
ISAKMP-peer=    HostA
Configuration=  Default-quick-mode
Local-ID=       Net-B
Remote-ID=      Net-A
```

These represent the sections referred to by the Phase 2 section above. They are the individual settings that ISAKMPD must use to talk between the two gateways for the particular connection.

- **Phase=2** is required because ISAKMPD code uses the same functions to authenticate Phase 1 and Phase 2. This is required for the VPN to work.
- **ISAKMPD-Peer=** is the name of Host section above. This means that we are talking to that particular peer to establish a Phase 2 connection. This is provided because you can have multiple sections to describe isakmp peers and connections.
- **Configuration=** refers to the section below that describes the standards by which this host and the particular peer for this connection must abide.
- **Local-ID=** refers to an IPsec-ID section below that describes our Private Network to the peer gateway. This is the portion that is passed so that the other gateway can set up the proper routing table that will transfer data over the VPN to our network.
- **Remote-ID=** refers to an IPsec-ID section below that describes what is supposed to be the remote Private Network to our host. This portion is interpreted to set up the proper routing tables that will transfer data from our Private Network over the VPN to the remote Private Network.

There is another tag that is supported here called **Flags=**. If you require this tag, read [isakmpd.conf\(5\)](#).

This is the IPsec-ID section. These entries need to exist in the `isakmpd.conf` files for both Host A and Host B. This example will set up 192.168.1.0/255.255.255.0 for Host A (which was connected to Net-A above) and 192.168.20.0/255.255.255.0 for Host B (Net-B above).

```
[Net-A]
ID-type=          IPV4_ADDR_SUBNET
Network=          192.168.1.0
Netmask=          255.255.255.0

[Net-B]
ID-type=          IPV4_ADDR_SUBNET
Network=          192.168.20.0
Netmask=          255.255.255.0
```

These two sections are in the **conf** file of each host. They are the sections referenced by the **Local-ID** and **Remote-ID** identifiers. They describe the routes that should be set up to allow traffic from one private network to another. **ID-type=** can be **IPV4_ADDR_SUBNET** or **IPV4_ADDR** (RFC2708 mentions more possible values. Currently only IPv4 is supported in the OpenBSD implementation. IPv6 may be supported in OpenBSD-current.)

Now, on both hosts, the sample file should read:

```
[Default-main-mode]
DOI=              IPSEC
EXCHANGE_TYPE=   ID_PROT
Transforms=      3DES-SHA
```

This section describes the requirements for the encryption methods of Phase 1 connections. The name reflects the value of *Configuration=* variable. As we can see here, we are stating our Domain of Interest which is IPSEC. The **EXCHANGE_TYPE** variable is set to **ID_PROT** for Phase 1, which identifies the protocols to be covered by this Authentication. **Transforms=** is the transform required (or assigned) for this exchange. In this case, this points to the section below in the configuration file that says we are receiving a packet encrypted with 3DES and a checksum verifiable with SHA. There are a bunch of different transforms defined inside the sample **VPN-east.conf**. These are provided because 3DES and SHA are not always supported across different platforms. For OpenBSD there should be no reason to change this for a basic setup. Feel free to create multiples of this section and change the transform. The only requirement is that you change the *Configuration=* variable.

```
[Default-quick-mode]
DOI=              IPSEC
EXCHANGE_TYPE=   QUICK_MODE
Suites=           QM-ESP-3DES-SHA-PFS-SUITE , QM-ESP-DES-MD5-PFS-SUITE
```

This section describes the requirements for the encryption of the data to be sent through the VPN and is referred to by *Configuration* above. Note the difference between this section and the Phase 1 equivalent just above is that the **EXCHANGE_TYPE** is **QUICK_MODE**. This is always the case for Phase 2. **Suites=** points to a IPsec Suite section describing the different encryption schemes available between the two hosts. There is much more to be said about ISAKMP and IPsec. By using the above basic descriptions you should be able to create a simple but solid VPN that cares and feeds itself. This is the bare *minimum* **isakmpd.conf** for both hosts here.

13.7.3 - Starting isakmpd

You may wish to use

```
# isakmpd -d -DA=90
```

the first time you decide to run this daemon. The daemon will not be running in daemon mode but as a regular (foreground) process. It will log everything to your terminal. To stop isakmpd and flush the routes, you need to kill the isakmpd process on each node, and run **ipsecadm flush**.

13.8 - How do I use isakmpd with X.509 certificates?

Setting up isakmpd to use certificates instead of pre-shared keys is not really that much harder in a big network with many untrusted peers than it would be with a small network. It may actually simplify configuration, and more importantly, makes key management much more flexible.

Generating certificates.

There is a good description of how to generate keys and certificates in the [README.PKI](#) file in the isakmpd source directory. You need to have a CA key, a corresponding CA X.509 certificate, one private key for each computer on the network that will use isakmpd and one X.509 certificate for each such key.

In order to be usable by isakmpd, the X.509 certificates need to have a Subject Alternative Name (subjectAltName) extension added to them. This extension describes the certificate holder's identity, and may be added by using either [certpatch\(8\)](#), or a custom openssl configuration file such as `/etc/ssl/x509v3.cnf`. The examples in [README.PKI](#) describe this process using IP addresses as subjectAltName identifiers.

Though IP addresses are the default mechanism for subjectAltName extensions, certpatch also supports using either a FQDN (Fully Qualified Domain Name) or a UFQDN (User FQDN). These can be very useful for mobile users. An example of an FQDN might be `www.openbsd.org`. An example of an UFQDN could be an email address, such as `Jorgen.Granstam@abc.se`. It should be noted that unlike IP-based identifiers, isakmpd performs no verification of the data presented in FQDN or UFQDN-based identification; the identifiers are treated simply as strings.

The following examples will use FQDNs as subjectAltNames.

To insert an FQDN subjectAltName into a certificate one would do something like this:

```
$ /usr/sbin/certpatch -t fqdn -i home.mysite.se -k ca.key originalcert.crt
newcert.crt
```

Here the `ca.key` is the private key of the CA, thus this can only be done by whoever has access to the CA private key. The (fictional) `home.mysite.se` is the FQDN to be inserted into the certificate. The `originalcert.crt` and `newcert.crt` filenames may be the same name in which case the original file will be overwritten by the new modified certificate.

These keys and certificates should then be moved to their appropriate directories. The default locations for the keys and certificates are as follows:

```
/etc/isakmpd/ca      public-key (PEM-format) certificates of trusted CAs
/etc/isakmpd/certs  trusted public-key (PEM-format) certificates (with subjectAltName extensions)
/etc/isakmpd/private Private Keys. These should have corresponding public keys in the cert directory, above
```

These locations may be overridden by values in the [X509-certificates] section of [isakmpd.conf\(5\)](#).

Note that if the CA infrastructure is to be trusted, the CA private key (`/etc/isakmpd/ca/ca.key`) should be kept in a safe place, should be of an appropriate bit length (i.e. 2048-bit), and should be encrypted with a strong passphrase.

Configuration of isakmpd

Lets now look at the `/etc/isakmpd/isakmpd.conf` configuration file. It was originally taken from the example file in [isakmpd.conf\(5\)](#) but has been heavily modified. I will also use a much shorter file here than the example file in the man page to make it easier to understand. I have also added some commenting (some comments are left as in the `isakmpd.conf(5)`) and changed some names. None of the domain names used here exist as far as I know.

Actually since everyone who reads this already has a working configuration for the pre-shared keys case (see previous section) there won't be many surprises in this file. I won't explain this in every detail, check `isakmpd.conf(5)` for descriptions of the parts I don't comment on.

Let's assume our setup looks something like this

```
one.mysite.se          one.worksite.se
 192.168.1.2--+      10.0.0.1====/=====10.0.0.2    +--192.168.2.2
                    | gw.mysite.se      gw.worksite.se |
                    +--192.168.1.1      192.168.2.1----+
two.mysite.se |
 192.168.1.3--+      | two.worksite.se
                    +--192.168.2.3
```

That is, two networks that should be connected using an IPsec tunnel over an otherwise insecure network. Ignore the fact that I am using IP addresses reserved for private Internets here (RFC1918), I have to use something. I won't explain how to use `isakmpd` in combination with NAT or similar (because I haven't tried that myself).

Now, let's look at the configuration file. This is the file for the security gateway `gw.mysite.se`:

```
# *****
# ***** Start of the gw.mysite.se isakmpd.conf *****
# *****

# A configuration sample for the isakmpd ISAKMP/Oakley (aka IKE) daemon.
[General]
Policy-File=          /etc/isakmpd/isakmpd.policy
Retransmits=         5
Exchange-max-time=   120
Listen-on=           10.0.0.1

# The name work-gw here is used just as a section name and a tag for
# use in this configuration file below and need not actually be the
# real hostname or domain name of the peer (but it could be). The IP
```

```

# address however needs to be correct. Phase 1, as you might already
# know, is to negotiate an ISAKMP security association (SA). There
# should of course be one IP and name for each peer we want to
# communicate with.
[Phase 1]
10.0.0.2=                work-gw

# Now phase 2 is negotiating IPsec SAs. As in phase 1, the name here
# is a section name to be used later. Actually, it can be a comma
# separated list of section names here. Thus if traffic from many
# networks (or individual hosts) should be forwarded through this
# tunnel, more section names would be added (and of course corresponding
# new sections further down).
[Phase 2]
Connections=            work-gw-my-gw

# Now, here are some parameters for the ISAKMP SA negotiations. Almost
# self documenting. The section name is from [Phase 1] above. The most
# interesting tag might be the ID tag. The ID tag is set to the name
# of the section where the identity information about this host that
# will be presented to connecting peers, can be found. If the ID tag
# is not available, isakmpd will assume that it will identify itself
# using the IP address. You might also notice that there is no longer
# any authentication tag here in this configuration. The authentication
# data is currently used only in the pre-shared key case.
[work-gw]
Phase=                  1
Transport=              udp
Local-address=          10.0.0.1                # Local address
Address=                10.0.0.2                # Peer address
ID=                     my-ID
Configuration=          Default-main-mode

# This is the identity data. ID-type may also be IPV4_ADDR (the
# default), IPV4_ADDR_SUBNET or UFQDN. The Name tag is used for
# FQDN and UFQDN, for IPV4_ADDR an Address tag would be used instead.
# For IPV4_ADDR_SUBNET a Network and a Netmask tag would be used.
[my-ID]
ID-type=                FQDN
Name=                   gw.mysite.se

# This is the section for the IPsec connection. The section name is
# from the list in the [Phase 2] section above. The ISAKMP-peer is,
# of course, the tag of our peer from section [Phase 1] above. The
# Local-ID and Remote-ID tags should be section names describing which
# packages should be forwarded over the IPsec tunnel to the remote
# network.
[work-gw-my-gw]
Phase=                  2
ISAKMP-peer=           work-gw
Configuration=          Default-quick-mode
Local-ID=               Net-west
Remote-ID=              Net-east

# Any packet originating from a computer on the network described
# here...
[Net-west]
ID-type=                IPV4_ADDR_SUBNET
Network=                192.168.1.0

```

```

Netmask=                255.255.255.0

# ... and with a destination matching the network described here,
# will be encrypted and forwarded over the IPsec tunnel to the remote
# system.
[Net-east]
ID-type=                IPV4_ADDR_SUBNET
Network=                192.168.2.0
Netmask=                255.255.255.0

# Main mode descriptions

# Here are the data for main mode. Using DES here for real purposes
# is not very smart since DES is no longer considered a secure
# encryption algorithm. 3DES is generally considered to have much better
# security since it has enough bits in the key to be considered secure.
# Transforms is a list of tags describing main mode transforms. In
# this example we have only one.
[Default-main-mode]
DOI=                    IPSEC
EXCHANGE_TYPE=          ID_PROT
Transforms=              3DES-MD5

# Certificates stored in PEM format
# This is important when using certificates. The CA certificates should
# be in the CA-directory (but not the CA private key of course).
# The Cert-directory should have at least the certificate for the
# local host but other certificates are also allowed. The private key
# should be the private key of the local host.
[X509-certificates]
CA-directory=            /etc/isakmpd/ca/
Cert-directory=          /etc/isakmpd/certs/
Private-key=             /etc/isakmpd/private/local.key

# Main mode transforms
#####

# Here is our main mode transform. The important thing here is to use
# RSA_SIG as authentication method when using certificates. It is the
# only method supported when using certificates so far. Commercial
# entities in the US will thus have to wait until September 2000 to
# use this due to the RSA patent. Luckily, I am not living in the US.
# Also important is the GROUP_DESCRIPTION tag. It must match the
# GROUP_DESCRIPTION tag in the Quick mode transforms further down.
# The Life tag here could possibly be modified. The LIFE_60_SECS might
# be shorter than necessary for normal use.

[3DES-MD5]
ENCRYPTION_ALGORITHM=   3DES_CBC
HASH_ALGORITHM=         MD5
AUTHENTICATION_METHOD=  RSA_SIG
GROUP_DESCRIPTION=      MODP_1024
Life=                    LIFE_60_SECS,LIFE_1000_KB

# Quick mode description
#####

[Default-quick-mode]
DOI=                    IPSEC
EXCHANGE_TYPE=          QUICK_MODE
Suites=                  QM-ESP-3DES-MD5-PFS-SUITE

```

```

# Quick mode protection suites
#####
# 3DES

[QM-ESP-3DES-MD5-PFS-SUITE]
Protocols=          QM-ESP-3DES-MD5-PFS

# 3DES

[QM-ESP-3DES-MD5-PFS]
PROTOCOL_ID=       IPSEC_ESP
Transforms=        QM-ESP-3DES-MD5-PFS-XF

# Quick mode transforms

# Don't forget. The GROUP_DESCRIPTION must match the GROUP_DESCRIPTION
# in main mode above. For forwarding packets between two networks (or
# from a host to a network) we use TUNNEL mode. Between two hosts we
# may also use TRANSPORT mode instead.
[QM-ESP-3DES-MD5-PFS-XF]
TRANSFORM_ID=      3DES
ENCAPSULATION_MODE= TUNNEL
AUTHENTICATION_ALGORITHM= HMAC_MD5
GROUP_DESCRIPTION= MODP_1024
Life=              LIFE_60_SECS

# As we know from the isakmpd.config manpage the LIFE_DURATION here is
# an offer value (60), a minimum acceptable value (45) and a maximum
# acceptable value. The isakmpd.conf example has this set to
# 600,450/720 instead. That might be a better value for normal use.
[LIFE_60_SECS]
LIFE_TYPE=         SECONDS
LIFE_DURATION=     60,45:72

[LIFE_1000_KB]
LIFE_TYPE=         KILOBYTES
LIFE_DURATION=     1000,768:1536

# *****
# ***** End of the gw.mysite.se isakmpd.conf *****
# *****

```

So far the configuration for the local system. The remote system is configured just the same way only opposite. Thus only the first part of the isakmpd.conf file differs. Let's just look at that first part of the isakmpd.conf file for the security gateway gw.worksite.se:

```

# *****
# ***** Start of the gw.worksite.se isakmpd.conf *****
# *****

[General]
Policy-File=       /etc/isakmpd/isakmpd.policy
Retransmits=      5
Exchange-max-time= 120
Listen-on=        10.0.0.2

[Phase 1]
10.0.0.1=         my-gw

[Phase 2]
Connections=      work-gw-my-gw

[my-gw]

```

```

Phase=                1
Transport=            udp
Local-address=        10.0.0.2          # Local address
Address=              10.0.0.1          # Peer address
ID=                  work-ID
Configuration=        Default-main-mode

[work-ID]
ID-type=              FQDN
Name=                 gw.worksite.se

[work-gw-my-gw]
Phase=                2
ISAKMP-peer=         my-gw
Configuration=        Default-quick-mode
Local-ID=             Net-east
Remote-ID=            Net-west

# *****
# ***** .. to be continued *****
# *****

```

Now that wasn't so hard, just a bit boring to read perhaps. A slightly more interesting part next.

The policy file.

Actually, the policy file might be slightly confusing for anyone who has not used it before, especially if things do not work as expected. The man-page [isakmpd.policy\(5\)](#) is not really that bad. It might perhaps be a little bit unclear in some parts but generally it's good.

The simplest possible working policy file would contain just a single line:

```
authorizer: "POLICY"
```

This basically means that there are no policy limitations on who would be allowed to connect. Thus not a very secure setup. The authorizer tag here means the one who has the authorization to decide the policy. The special authorizer "POLICY" has the ultimate and unlimited authority on policy. Any other authorizer must first be authorized by "POLICY" to have any authority here.

There can also be a set of conditions for what is allowed. The following policy thus would mean that only someone using the ESP protocol with some real encryption would be authorized (oh well, someone using DES would also be authorized here although DES could almost be considered snake oil today, it is left as an exercise for the reader to change this policy into not allowing DES either). Note that anyone who does encrypts with ESP would still be allowed.

```
authorizer: "POLICY"
conditions: app_domain == "IPsec policy" &&
            esp_present == "yes" &&
            esp_enc_alg != "null" -> "true";

```

It is also possible to "sublicense" authority to someone else (might be one or more entities). The simple case would be the pre-shared key case. In that case, anyone who knows the pre-shared passphrase is authorized. Thus:

```
authorizer: "POLICY"
licensees: "passphrase:something really secret"
conditions: app_domain == "IPsec policy" &&
            esp_present == "yes" &&

```

```
esp_enc_alg != "null" -> "true";
```

This would authorize anyone who knows this passphrase to connect and comply with the conditions (but remember that the passphrase must also be set in the Authentication tag in isakmpd.conf).

Nothing difficult so far. Now to the interesting stuff. First there can be many licensees, although all must be authorized by "POLICY". Further, authorized licensees can sublicense to other licensees. A licensee can be just a string in case it is further described in the policy file:

```
authorizer: "POLICY"
licensees: "subpolicyAH" || "subpolicyESP"
conditions: app_domain == "IPsec policy" -> "true";

authorizer: "subpolicyESP"
licensees: "passphrase:something more secret"
conditions: esp_present == "yes" -> "true";

authorizer: "subpolicyAH"
licensees: "passphrase:something really secret"
conditions: ah_present == "yes" -> "true";
```

And now to what everyone has been waiting for. Policy can also be sub-licensed or delegated to a key. In this case it is usually a X.509 certificate. The simple use of certificates would be to use them like the passphrases. Just insert individual users certificates in the policy file:

```
keynote-version: 2
comment: This is an example of a policy delegating to a key.
authorizer: "POLICY"
licensees: "x509-base64:\
    MIIBsTCCARoCAQAwDQYJKoZIhvcNAQEEBQAwITELMAkGA1UEBhMCc2UxEjAQBgNV\
    BAMTCUlLRUxBQIBDQTAeFw0wMDAxMjg0NzQyMTZaFw0wMTAxMjcxNzQyMTZaMCEX\
    CzA          This is would be a user certificate          AQEB\
    BQA          IUuz\
    eOW8P5UGJUH2JVkiaA2CTDryFf0CHYwd2P003dtVYw5RvET7XLMpRZiCcWtBdxneW\
    ct+016zUBP/cQMMl+KownxAUq9ezA8GvTyUWC97SOMogoVj/QR3FHmEjpUi3AgMB\
    AAEwDQYJKoZIhvcNAQEEBQADgYEALGShaAxHvGncev0iFnKrJI4x5T4v1aMPlad+\
    iWLV5q9H3wickVGN0NPerq0YLwx/VA9WaecYN8V+ALtNKYPuDiT11zvwE8GQeai\
    NuzgmQ9hh3GifEgN9VEiC3j4kTytonKr0Q+vTLM7xYzheOxvrtUeRrWz9Xs1KzHe\
    yiXHSU8="
conditions: app_domain == "IPsec policy" &&
            esp_present == "yes" &&
            esp_enc_alg != "null" -> "true";
```

Now, this is obviously a stupid idea if there are a lot of users. The certificates that isakmpd reads from the CA- and Certificate directories, and the certificates received from the peer is converted into pseudo credentials. Such certificates converted into pseudo credentials essentially would look something like:

```
authorizer: "x509-base64:\
    MIIBsTCCARoCAQAwDQYJKoZIhvcNAQEEBQAwITELMAkGA1UEBhMCc2UxEjAQBgI2\
    CzA This is would be the public key/certificate of the AQEB\
    BQA signer of the user certificate (i.e. the CA certificate)IUuz\
    eOW8P5UGJUH2JVkiaA2CTDryFf0CHYwd2P003dtVYw5RvET7XLMpRZiCcWtBdxneW\
    ct+016zUBP/cQMMl+KownxAUq9ezA8GvTyUWC97SOMogoVj/QR3FHmEjpUi3AgMB\
    AAEwDQYJKoZIhvcNAQEEBQADgYEALGShaAxHvGncev0iFnKrJI4x5T4v1aMPlad+\
    iWLV5q9H3wickVGN0NPerq0YLwx/VA9WaecYN8V+ALtNKYPuDiT11zvwE8GQeai\
    NuzgmQ9hh3GifEgN9VEiC3j4kTytonKr0Q+vTLM7xYzheOxvrtUeRrWz9Xs1KzHe\
    yiXHSU8="
```

```

licensees: "x509-base64:\
MIIBsTCCARoCAQAwDQYJKoZIhvcNAQEEBQAwITELMAkGA1UEBhMCc2UxEjAQBgNV\
BAMTCULLRUxBQiBDQTAeFw0wMDAxMjg5NzQyMTZaFw0wMTAxMjc5NzQyMTZaMCEX\
CzA          This is would be the key of the subject of the          AQEB\
BQA          certificate          IUuz\
eOW8P5UGJUH2JVkiaA2CTDryFf0CHYwd2P003dtVYw5RvET7XLMpRZiCcWtBdxneW\
ct+016zUBP/cQMMl+KownxAUq9ezA8GvTyUWC97SOMOgoVj/QR3FHmEjU3AgMB\
AAEwDQYJKoZIhvcNAQEEBQADgYEALGShaAxBvGncev0iFnKrJI4x5T4vlaMPlad+\
iWLV5q9H3wickVGN0NPerq0YLwx/VA9WaecYN8V+ALtNKYPuDiT1lzwvE8GQeaa\
NuzgmQ9hh3GifEgN9VEiC3j4kTytonKr0Q+vTLM7xYzheOxvrtUErRwZ9Xs1KzHe\
yiXHSU8="
conditions: app_domain == "IPsec policy" -> "true";

```

Now note that these are not authorized by "POLICY" and thus won't have any effect without a policy authorizing them somehow. Further, this showed what happens to certificates internally. The above credential is thus not seen in the policy file. However, it is possible to sublicense to such credentials. Remember sub-licensing above. It is thus possible to license all certificates that are signed by a certain CA by putting the CA certificate as a licensee to "POLICY":

```

keynote-version: 2
comment: This is an example of a policy delegating to a key.
authorizer: "POLICY"
licensees: "x509-base64:\
MIIBsTCCARoCAQAwDQYJKoZIhvcNAQEEBQAwITELMAkGA1UEBhMCc2UxEjAQBgNV\
BAMTCULLRUxBQiBDQTAeFw0wMDAxMjg5NzQyMTZaFw0wMTAxMjc5NzQyMTZaMCEX\
CzA          This would be the CA certificate          AQEB\
BQA          IUuz\
eOW8P5UGJUH2JVkiaA2CTDryFf0CHYwd2P003dtVYw5RvET7XLMpRZiCcWtBdxneW\
ct+016zUBP/cQMMl+KownxAUq9ezA8GvTyUWC97SOMOgoVj/QR3FHmEjU3AgMB\
AAEwDQYJKoZIhvcNAQEEBQADgYEALGShaAxBvGncev0iFnKrJI4x5T4vlaMPlad+\
iWLV5q9H3wickVGN0NPerq0YLwx/VA9WaecYN8V+ALtNKYPuDiT1lzwvE8GQeaa\
NuzgmQ9hh3GifEgN9VEiC3j4kTytonKr0Q+vTLM7xYzheOxvrtUErRwZ9Xs1KzHe\
yiXHSU8="
conditions: app_domain == "IPsec policy" &&
            esp_present == "yes" &&
            esp_enc_alg != "null" -> "true";

```

Thus the above policy is a simple example of a policy that delegates to a CA. Thus any user that has a certificate that is signed by the CA that has this certificate and otherwise comply to other conditions set by the policy and the configuration file would be authorized.

Almost secure...is not secure!

Now, to be really safe, this is not enough unfortunately. There are ways to attack a security gateway that is configured in this way. If you really don't want the details, skip a to the next section now. To everyone else, let's try to understand why this is not secure. It's not hard.

Consider what information one of the `isakmpds` has access to at this stage. From the configuration file, `isakmpd` knows which IP-address its peer will send from (in the [Phase 1] section). From the information it gets at the phase 1 negotiation it knows the ID that the peer presents itself with and the certificate it gets from the peer proves that the peer really have this ID. Looks fine so far?

Well, if the ID information were the IP (the default situation if we do not provide a phase 1 ID section) everything would be fine. The CA would have tied the IP to the cert and the IP in the configuration would be all information we need. It would be possible for an imposter to use the same IP from another computer in some cases (e.g. if both computers were on the same local network and the computer that usually have this IP is down for some reason). It should however not be possible for an imposter to be able to have a certificate and a corresponding private key that (falsely) proves that this IP belongs to the imposter.

If that ever happened, the imposter would have managed to either steal the private key from the real owner of the IP, or the imposter would have managed to fool the CA into issuing a certificate containing false information somehow. If any of these things happened,

then either the private key had not been protected well enough, the CA had failed to check the identity of the imposter well enough (or the ID info for the cert) or the CA private key had not been well enough protected. Since all these are prerequisites for security to work at all, none of these situations can be allowed to ever occur.

Now, in our example the situation is different. Here we actually have an FQDN in the certificate instead of an IP address. Since we still have an IP address in the [Phase 1] section this will result in a possible security problem. What now would happen during an ISAKMP phase 1 negotiation would be that we could check that the peer was sending from the expected IP (but as explained earlier that could possibly be forged in some situations). We could check that the ID our peer presents actually belongs to our peer. But what we can not check now is if that ID really is the ID we expect our peer to have, because isakmpd have never been told what ID that should be.

Someone now might say that the DNS system ties the IP to the FQDN for the host. That is true, however today's DNS system is not secure and can, under some circumstances, be fooled to give out false information (or, it could be subject to a denial of service (DoS) attack by an attacker, and the attacker's computer might be able to fake the DNS server's answer). Secure DNS will come in the future, but it is not here yet (at least most DNS servers are not secure yet), thus today, using DNS to check if the FQDN in the cert corresponds to the expected IP is no guarantee. In fact isakmpd does not check this with DNS. Even if DNS was secure, checking this would not help in the case of using an UFQDN.

Thus in the case of having an FQDN as ID, it could be possible for an attacker to get an own private key and having this key signed by the same CA that we use (but with the attacker's own FQDN, of course). Then launch a DoS attack on our peer so that it goes down (in fact, there are some flaws in the ISAKMP protocol itself that possibly could be used to launch a remote DoS against the peer and make it go down, although I don't know how sensitive isakmpd is to those attacks). Then the attacker could configure its own computer in the same way as our peer, connect it to our peers network and try to connect using its own ID, private key and certificate.

Since our own isakmpd has not been informed about what ID to our peer (and it is because the attacker is identically configured as our peer besides the certificate, ID and private key). Further our isakmpd can check that the certificate was signed by the same CA (but most CAs sign lots of certs, a cert might not be hard to get), and that the presented ID is the same as the ID in the cert. However it would not, with the configuration presented so far, check that this ID is the expected ID. Thus the attacker would be allowed to connect.

Preventing the attack.

The question now thus is, how can we inform isakmpd about what ID to expect? This is fortunately easy, and documented in the [isakmpd.policy\(5\)](#). We must do the check in the policy. Like this:

```
keynote-version: 2
comment: This is an example of a policy delegating to a key.
authorizer: "POLICY"
licensees: "x509-base64:\
    MIIBsTCCARoCAQAwDQYJKoZIhvcNAQEEBQAwITELMAkGA1UEBhMCc2UxEjAQBgNV\
    BAMTCULLRUxBQIBDQTAeFw0wMDAxMjg5NzQyMTZaFw0wMTAxMjcxNzQyMTZaMCEX\
    CzA          This would be the CA certificate          AQEB\
    BQA          IUuz\
    eOW8P5UGJUH2JVkiaA2CTDryFf0CHYwd2P003dtVYw5RvET7XLMprZiCcWtBdxneW\
    ct+016zUBP/cQMMl+KownxAUq9ezA8GvTyUWC97SOMogoVj/QR3FHmeJpUi3AgMB\
    AAEwDQYJKoZIhvcNAQEEBQADgYEALGShaAxHvGncev0iFnKrJI4x5T4vlaMPlad+\
    iWLV5q9H3wickVGN0NPerq0YLwx/VA9WaecYN8V+ALtNKYPuDiT1lzwvE8GQeaa\
    NuzgmQ9hh3GifEgN9VEiC3j4kTyt0nKr0Q+vTLM7xYzheOxvrtUERrWz9Xs1KzHe\
    yiXHSU8="
conditions: app_domain == "IPsec policy" &&
            esp_present == "yes" &&
            esp_enc_alg != "null" &&
            remote_id == "gw.worksite.se" -> "true";
```

Now only gw.worksite.se should be able to get an IPsec connection. More allowed IDs could easily be added by adding more alternative remote_id checks, e.g. by having conditions like these in the policy:


```

conditions: app_domain == "IPsec policy" &&
            esp_present == "yes" &&
            esp_enc_alg != "null" &&
            (remote_id == "gw.worksite.se" ||
             remote_id == "gw.whatsite.se") -> "true";

```

With this policy either of gw.worksite.se, gw.somesite.se or gw.whatsite.se could connect.

Some might say that is unfortunate that there has to be entire certificates inserted in the policy. It requires some work to reformat the certificates into the format in the policy and it makes the policy rather unreadable. If someone by mistake replaced a user certificate with the corresponding CA certificate somewhere in a complex policy it might cause unauthorized users to be allowed to connect in some cases and worse, it would not be easily detectable by reading through the policy file (since X.509 certificates are not in a human readable format).

Now, for the really bleeding edge people out there a solution to this problem is available. It is now possible to use the certificate Distinguished Name (DN) instead of a certificate in the policy (the corresponding certificate must of course be available from the certs or ca directories on disk so that isakmpd can find it). With this format the policy above might look like something like this instead:

```

keynote-version: 2
comment: This is an example of a policy delegating to a key.
authorizer: "POLICY"
licensees: "DN:/C=se/CN=IKELAB CA"
conditions: app_domain == "IPsec policy" &&
            esp_present == "yes" &&
            esp_enc_alg != "null" &&
            (remote_id == "gw.worksite.se" ||
             remote_id == "gw.somesite.se" ||
             remote_id == "gw.whatsite.se") -> "true";

```

Much more readable, isn't it? The information about what the exact DN for a certificate is can be found by looking at the certificate using the openssl utility. Something like:

```
$ openssl x509 -text < ca.crt
```

More complicated policy configurations are of course possible but this is a start anyway, and there is another example in the next section.

This should provide the necessary information about the certificate in ca.crt. Now this is good as long as we have a small or at least reasonably small policy. It is however still not too great if we have a gigantic site with lots of users that should be allowed to connect.

Multiuser configurations and/or centrally managed authorization.

Now, lets look at some really cool features of isakmpd. Previously we assumed that the expected peer was well known and had a static IP address. This is not always the case. Lots of people use dynamically assigned IPs or use many different computers. In other cases (like for a server) we might not know for sure who wants to connect.

Therefore one very nice feature of isakmpd is the ability to use a default tag instead of an IP in the [Phase 1] section, thus allowing isakmpd negotiations from any IP. This might thus look something like this:

```

[phase 1]
Default=          work-gw

```

First, it should be said that this configuration might not be secure from DoS attacks. As said before, there are some flaws in the

ISAKMP/IKE protocols. Anyway, using a default [phase 1] section also enable us to use a kind of "authorization certificates" instead.

Consider the case where we have a lot of authorized users but when we would not accept just any user. Like at a company. We would like company employees to be allowed to connect but nobody else. Now, imagine a big company where there might be thousands of employees. We might like them to all be able to connect from any computer (not only from within the company network) but everyone should not be allowed to do anything. It should now be possible to write a policy like this:

```
keynote-version: 2
authorizer: "POLICY"
licensees: "telnet@work" || "telnet@lab" || "pop3@work"
conditions: app_domain == "IPsec policy" &&
             esp_present == "yes" &&
             esp_enc_alg != "null" &&
             remote_id_type == "UFQDN" &&
             (remote_id == "telnet@worksite.se" ||
              remote_id == "pop3@worksite.se" ||
              remote_id == "telnet@lab.worksite.se") -> "true";

authorizer: "telnet@work"
licensees: "DN:/C=se/CN=IKELAB CA"
conditions: remote_id == "telnet@worksite.se" &&
             local_filter_type == "IPv4 address" &&
             local_filter_port == "23" &&
             local_filter == "192.168.002.003"

authorizer: "telnet@lab"
licensees: "DN:/C=se/CN=IKELAB CA"
conditions: remote_id == "telnet@lab.worksite.se" &&
             local_filter_type == "IPv4 address" &&
             local_filter_port == "23" &&
             local_filter == "192.168.002.002" -> "true";

authorizer: "pop3@work"
licensees: "DN:/C=se/CN=IKELAB CA"
conditions: local_filter_type == "IPv4 address" &&
             local_filter_port == "110" &&
             local_filter == "192.168.002.003" &&
             remote_id == "telnet@worksite.se" -> "true";
```

This might not be exactly how it should be. This is as far as I know completely untested (in fact, these filter conditions might not work at all as I expect). Also, a policy such as this one (in fact any with default as peer IP), would require rewrites of the `isakmpd.conf` file too. This would have some security implications too. Further, for this kind of connections where anyone should be allowed to connect, it would probably be desirable to log the DN of anyone who connected. `Isakmpd` does not yet support that to my knowledge. Also this probably could have other security implications. You are on your own, you have been warned. The basic idea should be clear anyway.

Just in case someone missed the really interesting possibilities this would have. If all computers using ISAKMP/IKE this way had a standard set of conditions for all services the users might like to use from remote, the CA could actually authorize users by just putting the right `subjectAltName` extensions in their certificates. Further, the expiration time for such certificates could be set to expire relatively often although the users would be able to download new reissued certificates when their current certificate is getting old. If the users misuse their authorizations, just stop reissuing the certificates and they won't get in more after it has expired. No need to change policy files on all computers just because an employee e.g. quits their job. The same scheme should work for other purposes than ISAKMP/IPsec too (including authorization for off-line systems!) although that would require special software. In any case, any organization doing this would probably want to be their own CA.

Multiuser configurations (mobile users) like these are possible with pre-shared keys too, but then it is required that `AGGRESSIVE` mode is used instead of `ID_PROT` mode since we then must be able to choose the right password phrase based on ID since we do not know that from what the IP is in this case (in `AGGRESSIVE` mode the ID is sent over at an earlier stage of the negotiation, but it is sent unencrypted, thus `AGGRESSIVE` mode is faster because it needs fewer message exchanges, but it also is a bit less secure since the ID is sent in clear).

13.9 - What IKE clients are compatible with isakmpd?

`isakmpd` is the ISAKMP/Oakley key management daemon that comes with OpenBSD. We suspect that it interoperates, at least partially, with most ISAKMP implementations, but the following have actually been tested. Note that some `isakmp` software out there is actually based on the OpenBSD `isakmp` daemon.

The following MS-Windows clients have been reported to be compatible:

- [Ashley Laurent](#) VPCOM VPN software
- [PGP VPN](#) software
- [Cisco](#) IRE client
- [Microsoft](#) Windows 2000, XP

The following gateways/routers have been reported to be compatible:

- [Cisco](#) IOS
- [Cisco](#) PIX
- [Intel](#) LanRover
- [Cendio](#) Fuego
- [KAME](#) for FreeBSD
- [FreeS/WAN](#) for Linux
- [Symantec](#) Raptor
- [Ericsson](#) eBox
- [F-Secure](#) VPN+
- [Teamware](#) TWISS
- [3com](#) Pathbuilder
- [Nortel](#) Contivity
- [CheckPoint](#) FW-1
- [Watchguard](#) Firebox III
- [Lucent](#) Access Point

13.10 - Troubleshooting IPsec/VPN

Your first tool for troubleshooting IPsec is [tcpdump\(8\)](#). Use `tcpdump` to look for several things.

First, if you are using `tcpdump` from OpenBSD, you have an enhanced version of `tcpdump` which can show some information about ESP and AH packets. If you are using `tcpdump` from OpenBSD 2.5 or on another operating system, chances are you have an older version that will simply show the protocol number for AH or ESP. (ESP is IP protocol 50, AH is 51)

- With `tcpdump`, look and see if traffic is using AH/ESP or cleartext. If your traffic is in cleartext, then your flows are set up incorrectly or your `isakmp` is not negotiating properly. Use [ping\(8\)](#) to generate simple traffic.

For instance, I have two hosts, 208.1.1.1 and 208.2.2.2. Logged in to 208.2.2.2, I am doing this:

```
# ping -c 3 208.1.1.1
PING esp.mil (208.1.1.1): 56 data bytes
64 bytes from 208.1.1.1: icmp_seq=0 ttl=255 time=190.155 ms
64 bytes from 208.1.1.1: icmp_seq=1 ttl=255 time=201.040 ms
64 bytes from 208.1.1.1: icmp_seq=2 ttl=255 time=165.481 ms
```

```
--- esp.mil ping statistics ---
3 packets transmitted, 3 packets received, 0% packet loss
round-trip min/avg/max = 165.481/185.558/201.040 ms
```

And in another session, I can see my encapsulated pings:

```
# tcpdump -ni fxp7 host 208.1.1.1
tcpdump: listening on fxp7
14:12:19.630274 esp 208.2.2.2 > 208.1.1.1 spi 0x00001000 seq 4535 len 116
14:12:19.813519 esp 208.1.1.1 > 208.2.2.2 spi 0x00001001 seq 49313 len 116
14:12:20.630277 esp 208.2.2.2 > 208.1.1.1 spi 0x00001000 seq 4536 len 116
14:12:20.832458 esp 208.1.1.1 > 208.2.2.2 spi 0x00001001 seq 49314 len 116
14:12:21.630273 esp 208.2.2.2 > 208.1.1.1 spi 0x00001000 seq 4537 len 116
^C
1831 packets received by filter
0 packets dropped by kernel
```

- ISAKMP runs on UDP port 500. If this is locked out through a firewall or packet filter, then you need to change it!

```
# Passing in ISAKMP traffic from the security gateways
pass in on ne0 proto udp from gatewB/32 port = 500 to gatewA/32 port = 500
pass out on ne0 proto udp from gatewA/32 port = 500 to gatewB/32 port = 500

# Passing in encrypted traffic from security gateways
pass in proto esp from gatewB/32 to gatewA/32
pass out proto esp from gatewA/32 to gatewB/32
```

- To turn on all useful debugging in isakmpd, start it as

```
# /sbin/isakmpd -d -DA=90
```

or (to skip the most detailed timer debug info)

```
# /sbin/isakmpd -d -DA=90 -D1=70
```

- You need to allow traffic which has been processed by IPsec from netB into your local firewalled netA.

```
# Passing in traffic from the designated subnets.
pass in on enc0 from netB/netBmask to netA/netAmask
```

- With tcpdump on OpenBSD, you can decode most cleartext parts of Internet Key Exchange sessions. Tcpcdump will also show AH payload data.
- Mount a /kern filesystem (if you don't use one by default already.)

```
# mkdir /kern; mount -t kernfs /kern /kern
```

In /kern, there is a table of current SA/SPIs, including which have flows (outgoing SAs) or not (incoming SAs). There are also traffic counters which you can use to see what traffic is going where.

- Finally, you can use [netstat\(1\)](#) to see your SAs.

```
$ netstat -rn -f encap
```

Routing tables

Encap:

Source	Port	Destination	Port	Proto	SA(Address/SPI/Proto)
0.0.0.0/32	0	192.168.99/24	0	0	208.1.1.1/00001000/50
0.0.0.0/32	0	208.1.1.1/32	0	0	208.1.1.1/00001000/50
208.1.2.0/24	0	192.168.99/24	0	0	208.1.1.1/00001000/50
208.1.2.0/24	0	208.1.1.1/32	0	0	208.1.1.1/00001000/50
208.1.5.0/24	0	192.168.99/24	0	0	208.1.1.1/00001000/50
208.1.5.0/24	0	208.1.1.1/32	0	0	208.1.1.1/00001000/50
208.2.2.2/32	0	192.168.99/24	0	0	208.1.1.1/00001000/50
208.2.2.2/32	0	208.1.1.1/32	0	0	208.1.1.1/00001000/50

- If all else fails, recompile your kernel with option `ENCDEBUG`. Then, set the `sysctl net.inet.ip.encdebug` to 1. Look in your `dmesg` for warnings or errors, and report them using [sendbug\(1\)](#) to the OpenBSD developers. Alternately, if you are not sure you have actually run into a bug, you may want to send a message to one of the [mailing lists](#).

13.11 - Related Documentation

IPsec is partially documented in the [vpn\(8\)](#) man page. There are various configuration templates in `/usr/share/ipsec/` directory which can also assist you. The manual pages for [enc\(4\)](#), [ipsec\(4\)](#), [ipsecadm\(8\)](#), [isakmpd\(8\)](#), [isakmpd.conf\(5\)](#) and [isakmpd.policy\(5\)](#) are detailed and can assist in setup and operation of IPsec.

Other links...

- [IETF IPsec Working Group](#)
- [SSH IPsec interoperability Test Node](#)
- [NIST IPsec Web Based Interoperability Tester](#)
- [A port of OpenBSD's IPsec to FreeBSD](#)
- [FreeS/WAN - IPsec for Linux](#)
- [OpenBSD 2.4 VPN Configuration Mini-FAQ](#)

And on to the RFCs...

- [RFC 1320](#) - The MD4 Message-Digest Algorithm
- [RFC 1321](#) - The MD5 Message-Digest Algorithm
- [RFC 1828](#) - IP Authentication using Keyed MD5
- [RFC 1829](#) - The ESP DES-CBC Transform
- [RFC 2040](#) - The RC5, RC5-CBC, RC5-CBC-Pad, and RC5-CTS Algorithms
- [RFC 2085](#) - HMAC-MD5 IP Authentication with Replay Prevention
- [RFC 2104](#) - HMAC: Keyed-Hashing for Message Authentication
- [RFC 2144](#) - The CAST-128 Encryption Algorithm
- [RFC 2202](#) - Test Cases for HMAC-MD5 and HMAC-SHA-1
- [RFC 2207](#) - RSVP Extensions for IPsec Data Flows
- [RFC 2268](#) - A Description of the RC2 Encryption Algorithm
- [RFC 2367](#) - PF_KEY Key Management API, Version 2
- [RFC 2401](#) - Security Architecture for the Internet Protocol (**IPsec**)
- [RFC 2402](#) - IP Authentication Header (**AH**)
- [RFC 2403](#) - The Use of HMAC-MD5-96 within ESP and AH
- [RFC 2404](#) - The Use of HMAC-SHA-1-96 within ESP and AH
- [RFC 2405](#) - The ESP DES-CBC Cipher Algorithm With Explicit IV
- [RFC 2406](#) - IP Encapsulating Security Payload (**ESP**)

- [RFC 2407](#) - The Internet IP Security Domain of Interpretation for ISAKMP
- [RFC 2408](#) - Internet Security Association and Key Management Protocol (**ISAKMP**)
- [RFC 2409](#) - The Internet Key Exchange (**IKE**)
- [RFC 2410](#) - The NULL Encryption Algorithm and Its Use With IPsec (ha ha...)
- [RFC 2411](#) - IP Security Document Roadmap
- [RFC 2412](#) - The OAKLEY Key Determination Protocol
- [RFC 2451](#) - The ESP CBC-Mode Cipher Algorithms
- [RFC 2631](#) - Diffie-Hellman Key Agreement Method
- [RFC 2709](#) - Security Model with Tunnel-mode IPsec for NAT Domains

[\[FAQ Index\]](#) [\[To Section 12 - For Advanced Users\]](#) [\[To Section 14 - Using disks in OpenBSD\]](#)



www@openbsd.org

\$OpenBSD: faq13.html,v 1.74 2003/04/03 15:12:53 nick Exp \$



[\[FAQ Index\]](#) [\[To Section 13 - IPsec\]](#)

14 - Disk Setup

Table of Contents

- [14.1 - Using OpenBSD's disklabel](#)
 - [14.2 - Using OpenBSD's fdisk](#)
 - [14.3 - Adding extra disks in OpenBSD](#)
 - [14.4 - How to swap to a file](#)
 - [14.5 - Soft Updates](#)
 - [14.6 - When I boot after installation of OpenBSD/i386, it stops at "Using Drive: 0 Partition 3".](#)
 - [14.7 - What are the issues regarding large drives with OpenBSD?](#)
 - [14.8 - Installing Bootblocks - i386 specific](#)
 - [14.9 - Preparing for disaster: Backing up and Restoring from tape.](#)
 - [14.10 - Mounting disk images in OpenBSD](#)
 - [14.11 - Help! I'm getting errors with PCIIDE!](#)
 - [14.12 - Forcing DMA access for IDE disks](#)
 - [14.13 - RAID options with OpenBSD](#)
-

Using OpenBSD's disklabel

Table of Contents

- [What is disklabel?](#)
- [disklabel during the OpenBSD install](#)
- [Common disklabel uses.](#)

What is disklabel?

First be sure to read the [disklabel\(8\)](#) man page.

Disklabels are created to allow an efficient interface between your disk and the disk drivers contained within the kernel. Labels hold certain information about your disk, like your drive geometry and information about your filesystems. This is then used by the bootstrap program to load the drive and to know where filesystems are contained on the drive. Labels are also used in conjunction with the filesystems to create a more efficient environment. You can read more in-depth information about disklabel by reading the [disklabel\(5\)](#) man page.

As an additional gain, using disklabel helps overcome architecture limitations on disk partitioning. For example, on i386, you can only have 4 primary partitions. (Partitions that other operating systems, such as Windows NT or DOS can see.) With [disklabel\(8\)](#), you use one of these 'primary' partitions to store *all* of your OpenBSD partitions (eg. 'swap', '/', '/usr' and '/var'). And you still have 3 more partitions available for other OSs!

disklabel during OpenBSD's install

One of the major parts of OpenBSD's install is your initial creation of labels. This comes (for i386 users) directly after using [fdisk\(1\)](#). During the install you use disklabel to create your separate labels which will contain your separate mountpoints. During the install, you can set your mountpoints from within [disklabel\(8\)](#), but this isn't completely necessary considering you will be prompted later to confirm your choices. But it does make your install go just a little smoother.

Since this is during the install you won't have any existing labels, and they will need to be created. The first label you will create is the label 'a'. This label SHOULD be your where / will be mounted. You can see recommended partitions that should be created and their sizes by reading [FAQ 4, Space Needed](#). For servers it is recommended that you create at least these label's separately. For desktop users creating one mountpoint at / will probably suffice. When initially creating your root

partition ('a' label), keep in mind that you will need SOME space left for your swap label. Now that the basics have been explained, here is an example of using disklabel during an install. In this first example it is assumed that OpenBSD will be the only operating system on this computer, and that a full install will be done.

If this disk is shared with other operating systems, those operating systems should have a BIOS partition entry that spans the space they occupy completely. For safety, also make sure all OpenBSD file systems are within the offset and size specified in the 'A6' BIOS partition table. (By default, the disklabel editor will try to enforce this). If you are unsure of how to use multiple partitions properly (ie. separating /, /usr, /tmp, /var, /usr/local, and other things) just split the space into a root and swap partition for now.

```
# using MBR partition 3: type A6 off 63 (0x3f) size 4991553 (0x4c2a41)
```

Treating sectors 63-16386300 as the OpenBSD portion of the disk.
You can use the 'b' command to change this.

Initial label editor (enter '?' for help at any prompt)

```
> d a
> a a
offset: [63] <Enter>
size: [16386237] 64M
Rounding to nearest cylinder: 131040
FS type: [4.2BSD] <Enter>
mount point: [none] /
fragment size: [1024] <Enter>
block size: [8192] <Enter>
cpg: [16] <Enter>
> a b
offset: [131103] <Enter>
size: [16255197] 64M
Rounding to nearest cylinder: 131040
FS type: [swap] <Enter>
```

At this point we have created a 64M root partition mounted at /, and a 64Meg swap partition. Notice that the offset starts at sector 63. This is what you want. When it comes to the size, disklabel will show your size in sectors, however, you don't need to enter sizes in the same format. Like the example above you can enter sizes in the manner of *64 Megabytes = 64M* and *2 Gigabytes = 2G*. Disklabel will then round to the nearest cylinder. In the example above you will also notice that disklabel assumes that label 'b' will be a swap. This is a correct assumption as the GENERIC kernel is set to look for swap on label 'b', and you should just follow this guideline and use 'b' as your swap area.

The next example will take you through the creation of two more labels. This means that it's not a complete install, as the size of these won't be enough to install OpenBSD to its fullest. Showing the creation of all the partitions would just be repetitive.

```
> a d
offset: [262143] <Enter>
size: [16124157] 64M
Rounding to nearest cylinder: 131040
FS type: [4.2BSD] <Enter>
mount point: [none] /tmp
fragment size: [1024] <Enter>
block size: [8192] <Enter>
cpg: [16] <Enter>
> a e
offset: [393183] <Enter>
size: [15993117] 64M
Rounding to nearest cylinder: 131040
FS type: [4.2BSD] <Enter>
mount point: [none] /var
fragment size: [1024] <Enter>
block size: [8192] <Enter>
cpg: [16] <Enter>
```

In the above example, there are two things you might notice. One being that the offset is automatically figured out for you to be the next in order. When doing an install of this sort, you won't need to mess with changing the offsets at all. Another difference you might notice will be that label 'c' has been skipped. This is done for a reason, which is that label 'c' is a label that represents the whole disk. For this reason you shouldn't deal with label 'c' in any way.

Once all your labels have been created all that's left to do is write the labels to disk, and move on in the installation process. To write everything and quit disklabel (and continue with the install) do:

```
> w
> q
```


NOTE - For users with large drives. If your bios isn't able to support a drive of that size OpenBSD cannot support it either. Otherwise OpenBSD should be able to handle your drive just fine. If you are in a situation where your bios doesn't support your drive, you can try Maxtor EZ-Drive or other similar overlay product.

Common uses for disklabel(8)

Once your system is installed, you shouldn't need to use disklabel too often. But some times you will need to use disklabel when adding, removing or restructuring your disks. One of the first things you will need to do is view your current disklabel. To do this, simply type:

```
# disklabel wd0 >----- Or whatever disk device you'd like to view

# using MBR partition 3: type A6 off 64 (0x40) size 16777152 (0xffffc0)
# /dev/rwd0c:
type: ESDI
disk:
label: TOSHIBA MK2720FC
flags:
bytes/sector: 512
sectors/track: 63
tracks/cylinder: 16
sectors/cylinder: 1008
cylinders: 2633
total sectors: 2654064
rpm: 3600
interleave: 1
trackskew: 0
cylinderskew: 0
headswitch: 0      # milliseconds
track-to-track seek: 0 # milliseconds
drivedata: 0

16 partitions:
#      size  offset  fstype  [fsize bsize  cpg]
a:  2071440  65583  4.2BSD  1024  8192   16 # (Cyl.  65*- 2120)
b:   65520    63    swap                # (Cyl.  0*- 65)
c:  2654064    0  unused                # (Cyl.  0 - 2632)
j:   512001  2137023  4.2BSD  1024  8192   16 # (Cyl.  2120*- 2627*)
```

The above command simply allows you to view the existing disklabel, and assuring that you dont mess anything up. (Which we all need sometimes.) But to be able to make changes you must use the -E option with disklabel like so:

```
# disklabel -E wd0
```

This will bring you to a prompt, the same as the one that you used during the OpenBSD install. Probably the single most important command at this prompt is '?'. This will give you a list of possible options pertaining to disklabel. You can even view the entire [disklabel\(8\)](#) man page with the 'M' command. From this prompt, you will do all of your adding, deleting and changing of partitions. For additional information read the [disklabel\(8\)](#) man page.

14.2 - Using fdisk

First be sure to check the fdisk man page. [fdisk\(8\)](#)

Fdisk is a program to help with the maintenance of your partitions. This program is used at install time to set up your OpenBSD partition (this partition can contain several labels, each with filesystems/swap/etc.). It can divide space on your drives and set one active. This program will usually be used in Single User Mode (boot - s). Fdisk also sets the MBR on your various hard disks.

For installation purposes, most times you'll only need **ONE** OpenBSD partition, and then using disklabel to put a swap and a filesystem on it.

To just view your partition table using fdisk, use:

```
# fdisk fd0
```

Which will give an output similar to this:

```
Disk: fd0      geometry: 80/2/18 [2880 sectors]
```

```

Offset: 0      Signatures: 0xAA55,0x0
      Starting      Ending
#  id  cyl  hd sec -  cyl  hd sec [  start -  size]
-----
*0: A6   0   0   1 -   79   1  18 [  0 -  2880] OpenBSD
  1: 00   0   0   0 -    0   0   0 [  0 -    0] unused
  2: A7   0   0   2 -   79   1  18 [  1 -  2879] NEXTSTEP
  3: 00   0   0   0 -    0   0   0 [  0 -    0] unused

```

In this example we are viewing the fdisk output of floppy drive. We can see the OpenBSD partition (A6) and its size. The * tells us that the OpenBSD partition is a 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
      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>

```

It is perfectly safe in fdisk to go in and explore, just make sure to answer **N** to saving the changes and ***DON'T*** use the **write** command.

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.
- **disk** Display the current drive geometry that fdisk has probed. 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 BIOS 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.
- **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.

14.3 - 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 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
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! For normal operations, `fsize` should always be 1024, `bsize` should always be 8192, and `cpg` should always be 16. `Fstype` 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, if you have a disk big enough, 96G for 96 gigs!) Unfortunately, the -E mode uses the BIOS disk geometry, not the real disk geometry, and often times the two are not the same. To get around this limitation, type 'g d' for 'geometry disk'. (Other options are 'g b' for 'geometry bios' and 'g u' for geometry user, or simply, what the label said before `disklabel` made any changes.)

That was a lot. But you are not finished. Finally, you need to create the filesystem on that disk using [newfs\(8\)](#).

```
# newfs sd2a
```

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/sd2a /u
```

Finally, add it to [/etc/fstab\(5\)](#).

```
/dev/sd2a /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/sd2a (your freshly formatted partition.) Example:

```
/dev/sd2a /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!

14.4 - How to swap 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 `csch's unlimit\(1\)`, or `sh's ulimit\(1\)`.)

Swapping to a file doesn't require a custom built kernel, although that can still be done, this faq will show you how to add swap space both ways.

Swapping to a file.

Swapping to a file is easiest and quickest way to get extra swap space setup. 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 large.

```
$ 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
```

Swapping via a vnode device

This is a more permanent solution to adding more swap space. To swap to a file permanently, first make a kernel with `vnd0c` as swap. If you have `wd0a` as root filesystem, `wd0b` is the previous swap, use this line in the kernel configuration file (refer to compiling a new kernel if in doubt):

```
config          bsd          root on wd0a swap on wd0b and vnd0c dumps on wd0b
```

After this is done, the file which will be used for swapping needs to be created. You should do this by using the same command as in the above examples.

```
$ 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)
```

Now your file is in place, you need to add the file to you `/etc/fstab`. Here is a sample line to boot with this device as swap on boot.

```
$ cat /etc/fstab
/dev/wd0a / ffs rw 1 1
/dev/vnd0c none swap sw 0 0
```

At this point your computer needs to be rebooted so that the kernel changes can take place. Once this has been done it's time to configure the device as swap. To do this you will use [vnconfig\(8\)](#).

```
$ sudo vnconfig -c -v vnd0 /var/swap
vnd0: 33554432 bytes on /var/swap
```

Now for the last step, turning on swapping to that device. We will do this just like in the above examples, using `swapctl(8)`. Then we will check to see if it was correctly added to our list of swap devices.

```
$ sudo swapctl -a /dev/vnd0c
$ swapctl -l
Device      512-blocks      Used    Avail Capacity  Priority
swap_device  65520           8      65512    0%      0
/dev/vnd0c   65536           0      65536    0%      0
Total       131056          8     131048    0%
```

14.5 - Soft Updates

Over the last few years Kirk McKusick has been working on something called "Soft Updates". This is based on an idea proposed by Greg Ganger and Yale Patt that imposing a partial ordering on the buffer cache operations would permit the requirement for synchronous writing of directory entries to be removed from the FFS code. Thus, a large performance increase of disk writing performance.

As Soft Updates are still in development as a whole, an [fsck\(8\)](#) is still needed after the computer is abruptly turned off without a clean shutdown sequence, but this will be remedied in future versions.

More internals and details about Soft Updates can be found in the papers of [Ganger and Patt](#) and from [McKusick](#).

To use Soft Updates, your kernel must have

option FFS_SOFTUPDATES

compiled in, this is already in place on GENERIC.

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 spare 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.

14.6 - When I boot after installation of OpenBSD/i386, it stops at "Using Drive: 0 Partition: 3" - i386 specific.

This isn't actually an error message by itself, it is the boot loader in the MBR telling you which drive and partition it is about to boot from. The problem is, the boot process stopped here.

There are two common reasons why this may happen: an incompatibility between the BIOS and the OpenBSD MBR, or a drive geometry problem. An example of a drive geometry problem would be if you were to move a drive from one computer to another, update a BIOS or change a BIOS setting on a computer, though it is also reported that it can happen for unknown reasons during install.

Note: As of OpenBSD 3.1, the BIOS compatibility problem should be resolved on virtually all computer systems. However, it is still possible to have a drive geometry problem, so it is still possible to have the system hang after this message.

To fix the BIOS compatibility issue, you must replace the boot loader with one that is compatible with your system. Fortunately, boot loaders are easy to come by.

Installing the BootEasy boot loader:

This requires booting your system. As you can't boot off the hard disk directly, we have to use a boot floppy or CD-ROM to start the boot process. When the system gets to the 'boot>' prompt, redirect it to boot from the hard disk:

```
reading boot.....
```

```
probing pc0 com0 com1 pci mem [639k 79m a20=on]
disk hd0 fd0
>> OpenBSD BOOT 1.28
boot> boot hd0a:/bsd
```

This command redirects the boot process to the file /bsd on the 'a' partition of the first hard drive, allowing your system to boot.

Once the system is booted, you need to install the BootEasy boot loader. The file itself can be found on both the CD-ROM and the FTP sites, in the directory 3.0/tools/booteasy/Boot.bin, and can be installed using the following [fdisk\(8\)](#) command:

```
# fdisk -i -f /mnt/3.0/tools/booteasy/Boot.bin wd0
```

This assumes you have the 3.0 CD mounted on /mnt and that you are using an IDE drive. You may need to change this depending upon where Boot.bin is and what kind of hard disk you have (a SCSI drive would typically be 'sd0'). **Note: do NOT do this if your OpenBSD partition is not the entire disk!** Initializing the MBR this way creates one OpenBSD partition spanning the entire disk and deletes all other partitions, which is rarely good.

BootEasy has another feature that might interest you even if you have no problem with the default loader: It is capable of selecting a boot partition at startup -- it will prompt you for which partition to boot from, and will boot the active partition if no other choice is made. This might be very useful should you have multiple OS's on one disk. Installing BootEasy this way is done from MS-DOS with the BOOTINST.EXE program found in the BootEasy directory on the CD-ROM and FTP servers.

Installing the MS-DOS boot loader:

Boot from a Windows 9x or DOS v6 boot disk with FDISK.EXE on it. Once the system is booted to an MS-DOS prompt, enter the following:

```
A:\>fdisk /mbr
```

You should see a brief disk access, and then a command prompt should return, with NO message of any kind. "Bad command or file name" means the disk you used did not have FDISK.EXE on it. If you do this properly, the "Using ..." message will be removed, as it replaces the very code that produces that message. If you had a BIOS compatibility issue, it will now be gone, reboot, your OpenBSD install should come right up.

It has been reported that this also works with FreeDOS.

The OS-BS Boot Loader:

Another boot loader, OS-BS, is included with the OpenBSD CD-ROMs and available on the FTP sites in 2.9/tools/osbs135.exe. The OS-BS web page is at <http://www.prz.tu-berlin.de/~wolf/os-bs.html>.

LILO:

The Linux LILO program can also be used. For details, see [INSTALL.linux](#)

Avoiding the problem:

Relatively few machines have the BIOS compatibility problem, but if you are setting up a machine that you know has this problem, it is fairly easy to avoid it. The only time the OpenBSD loader is installed on your system during a normal install is when you say 'Y' to the 'Use entire disk for OpenBSD' question. If you answer 'N' and manually create the OpenBSD disk partition, it will not replace the boot loader unless you use the 'reinit' or 'update' commands of fdisk. This, of course, assumes that your drive STARTED with some kind of boot loader -- if it doesn't (new, blank drive, drive pulled from another platform), you will have to install one before the system will boot.

Fixing a drive geometry problem:

Ideally, you would want to avoid this problem by maintaining the same drive geometry, not by fixing it, however sometimes you can't avoid it. An example would be moving a large drive from an old computer which didn't support LBA geometry to a new machine which insists upon using LBA.

Start your machine using a boot disk or CD-ROM, as indicated above. Log in as root, and execute the following commands:

```
# cp /usr/mdec/boot /boot
# /usr/mdec/installboot -v /boot /usr/mdec/biosboot wd0
```

Reboot, your system should come right up.

[installboot\(8\)](#) installs and configures the partition boot loader, [biosboot\(8\)](#), which loads [boot\(8\)](#). [boot\(8\)](#) is the module which loads the kernel into RAM. [biosboot\(8\)](#) has a table within it that points to the physical location (according to the system's BIOS) of [boot\(8\)](#). If you do anything which changes the BIOS's perception of the location of [boot\(8\)](#), you must re-run [installboot\(8\)](#) as above to reinitialize the table pointing to [boot\(8\)](#).

See [Install Boot](#) for more info.

14.7 - What are the issues regarding large drives with OpenBSD?

OpenBSD has support for file systems of sizes much larger than any currently or soon to be available hard disks, however there are limitations on some interfaces which are smaller than the theoretical maximum of OpenBSD. In the case of IDE drives, the limit is 128GB, the limit of the currently popular ATA interface. Note the next generation of ATA drives, those with capacities greater than 128G (1G=2³⁰ here, not 1,000,000,000, so drive manufacturers will often call this 137G), are not supported by OpenBSD 3.1 and earlier.

Unfortunately, the full ability of the OS isn't available until AFTER the OS has been loaded into memory, and the booting process introduces limits of its own. The boot process has to utilize (and is thus limited by) the system's boot ROM. The OpenBSD i386 boot loaders ([biosboot\(8\)](#) and [boot\(8\)](#)) also have their own internal 8G limitation, from an older BIOS limit.

For this reason, the entire /bsd file (the kernel) must be located on the disk within the boot ROM addressable area, or within the first 8G of the disk, whichever is smaller. This means that on some older i386 systems, the root partition must be completely within the first 504M, but for most newer computers, the root partition may be anywhere within the first 8G.

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 "bad magic"

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, /bsd must be within the boot ROM's supported range. To play it safe, the rule is simple:

The entire root partition must be within the computer's BIOS (or boot ROM) addressable space or within the first 8G, whichever is smaller. There are no ways around this at this time. Trust us. You *must* follow this rule.

This is another good reason to [partition your hard disk](#), rather than using one large partition.

14.8 - Installing Bootblocks - i386 specific

Older versions of MS-DOS can only deal with disk geometries of 1024 cylinders or less. Since virtually all modern disks have more than 1024 cylinders, most SCSI BIOS chips (which come on the SCSI controller card) and IDE BIOS (which is part of the rest of the PC BIOS) have an option (sometimes the default) to "translate" the real disk geometry into something that fits within MS-DOS' ability. However, not all BIOS chips "translate" the geometry in the same way. If you change your BIOS (either with a new motherboard or a new SCSI controller), and the new one uses a different "translated" geometry, you will be unable to load the second stage boot loader (and thus unable to load the kernel). (This is because the first stage boot loader contains a list of the blocks that comprise /boot in terms of the original "translated" geometry). If you are using IDE disks, and you make changes to your BIOS settings, you can (unknowingly) change its translation also (most IDE BIOS offer 3 different translations). To fix your boot block so that you can boot normally, just put a boot floppy in your drive (or use a bootable CDROM) and at the boot prompt, type "b hd0a:/bsd" to force it to boot from the first hard disk (and not the floppy). Your machine should come up normally. You now need to update the first stage boot loader to see the new geometry (and re-write the boot block accordingly).

Our example will assume your boot disk is sd0 (but for IDE it would be wd0, etc.):

```
# cd /usr/mdec; ./installboot /boot biosboot sd0
```

If installboot complains that it is unable to read the BIOS geometry, at the boot> prompt you may issue the "machine diskinfo" (or "ma di" for short) command to print the information you need. Feed the "heads" and "secs" values to installboot's -h and -s flags, respectively, so that the modified installboot command is the following:

```
# cd /usr/mdec; ./installboot -h <heads> -s <secs> /boot biosboot sd0
```

If a newer version of bootblocks are required, you will need to compile these yourself. To do so simply:

```
# cd /sys/arch/i386/stand/
# make && make install
# cd /usr/mdec; cp ./boot /boot
# ./installboot /boot biosboot sd0 (or whatever device your hard disk is)
```

14.9 - 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.

This information will assist you in using the standard [dump\(8\)/restore\(8\)](#) utilities provided with OpenBSD. A more advanced backup utility called "Amanda" is also available for backing up multiple servers to one tape drive. In most environments [dump\(8\)/restore\(8\)](#) is enough. However, if you have a need to backup multiple machines to one tape, Amanda might be worth investigating in the future.

The device examples in this document are for a configuration that uses both SCSI disks and tape. In a production environment, SCSI disks are recommended over IDE due to the way in which they handle bad blocks. That is not to say this information is useless if you are using an IDE disk or other type of tape drive, your device names will simply differ slightly. For example sd0a would be wd0a in an IDE based system.

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 SCSI fixed disk 0, partition a. The /usr filesystem resides on sd0h, which indicates SCSI 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 [fdisk FAQ](#) 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 [disklabel FAQ](#) 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 remounted 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.

14.10 - 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 svnd0 /tmp/ISO.image
# mount -t cd9660 /dev/svnd0c /mnt
```

Notice that, since this image is a CD image you must specify type of `cd9660` when mounting it. This is true, no matter what type, e.g. you must use type `ffs` when mounting disk images.

To unmount the image use the following commands.

```
# umount /mnt
# vnconfig -u svnd0
```

For more information, refer to the [vnconfig\(8\)](#) man page.

14.11 - Help! I'm getting errors with PCIIDE!

PCI IDE DMA is unreliable with many combinations of hardware. Until recently, most "mainstream" operating systems that claimed to support DMA transfers with IDE drives did not ship with that feature active by default.

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.

If OpenBSD does not successfully downgrade, or the process causes your machine to lock hard, please send in a [bug report](#).

14.12 - Forcing DMA access for IDE disks

With the PCI IDE code, your chipset may not be known. If so, you will get a message like:

```
pciide0: DMA, (unused)
```

If you get this message, you can try and force DMA mode by using 'flags 0x0001' on your `pciide` entry in your kernel config file. That would look something like

this:

```
pciide* at pci ? dev ? function ? flags 0x0001
```

After doing this, the pciide code will try to use DMA mode regardless of whether or not it actually knows how to do so with your chipset. If this works, and your system makes it through fsck and startup, it is likely that this will work for good. If this does not work, and the system hangs or panics after booting, then you can't use DMA mode (yet, until support is added for your chipset). Note that if you find the documentation for your PCI-IDE controller's chipset, this is a good start to fully supporting your chipset within the PCI-IDE code. You can look on the manufacturer's website or call them. If your PCI-IDE controller is part of your motherboard, figure out who manufactures the chipset and pursue their resources!

Note that you will know that DMA support has been enabled if you see this message:

```
cd0(pciide0:1:0): using PIO mode 3, DMA mode 1
```

This means that pciide0, channel 1, drive 0 (which appears to be an ATAPI CD-ROM) is using DMA data transfers.

14.13 - RAID options for OpenBSD

RAID (Redundant Array of Inexpensive Disks) gives an opportunity to use multiple drives to give better performance, capacity and/or redundancy than one can get out of a single drive alone. While a full discussion of the benefits and risks of RAID are outside the scope of this article, there are a couple points that are important to make here:

- RAID has nothing to do with backup.
- By itself, RAID will not eliminate down-time.

If this is new information to you, this is not a good starting point for your exploration of RAID.

Software Options

OpenBSD includes RAIDframe, a software RAID solution. Documentation for it can be found in the following places:

- [FAQ 11, RAID](#)
- [RAIDframe Homepage](#)
- [man page for raidctl\(8\)](#)
- [man page for raid\(4\)](#)

Starting with OpenBSD 3.1, the root partition can now be directly mirrored by OpenBSD using the "Autoconfiguration" option of RAIDframe. Previous releases of OpenBSD could not have the root partition mirrored this way.

Hardware Options

Many OpenBSD [platforms](#) include support for various hardware RAID products. The options vary by platform, see the appropriate hardware support page (listed [here](#)).

Another option available for many platforms is one of the many products which make multiple drives act as a single IDE or SCSI drive, and are then plugged into a standard IDE or SCSI adapter. These devices can work on virtually any hardware platform that supports either SCSI or IDE.

Some manufacturers of these products:

- [Arco](#)
- [Maxtronic](#)
- [Infortrend](#)

(Note: these are just products that OpenBSD users have reported using -- this is not any kind of endorsement, nor is it an exhaustive list.)

Non-Options

An often asked question on the [mail lists](#) is "Is the Promise or HighPoint IDE RAID controller supported?". The answer is "No". These cards and chips are not true hardware RAID controllers, but rather BIOS-assisted boot of a software RAID. As OpenBSD already supports software RAID in a hardware-independent way, there

isn't much desire among the OpenBSD developers to implement special support for these cards.

[\[FAQ Index\]](#) [\[To Section 13 - IPsec\]](#)



www@openbsd.org

\$OpenBSD: faq14.html,v 1.82 2003/04/04 17:49:08 nick Exp \$