You've installed it. Now what? Packages!

Peter N. M. Hansteen

A freshly installed machine nice, but it's when you start using the package tools that the real visitas open. Read on for a kickstart on packages.

Installing OpenBSD is easy, and takes you maybe 20 minutes. Most articles and guides you find out there will urge you to take a look at the files in `/etc/` and explore the man pages to make the system do what you want. With a modern BSD, the base system is full featured enough that you can in fact get a lot done right away just by editing the relevant files and perhaps starting or restarting one or more services. If all you want to do is set up something like a gateway for your network with basic-to-advanced packet filtering, everything you need is already there in the basic install.

Then again, all the world is not a firewall, and it is likely you will want to use, for example, a web browser other than the venerable lynx or editing tools that are not vi or mg. That's where packages and package systems come in. I will skip a little ahead of myself and make a confession: The machine I am writing this piece on reports that it has some 260 packages installed.

Before we move on to the guts of this article, some ceremonial words of advice: If you are new to OpenBSD or it is your first time in a while on a freshly installed system, you could do a lot worse than spending a few minutes reading man afterboot. That man page serves as a handy checklist of things you should at least take a peek at to ensure that your system is in good working order.

Some packages will write important information, such as strings or stanzas to put in your `rc.conf.local`, `rc.local` or `sysctl.conf` files, to your terminal. If you are not totally confident what to do after the package install finishes, it may be a good idea to run your ports and packages installs in a script session. See man script for details.

When dinosaurs roamed the Earth...

The story of the ports and packages goes back to the early days of free software when we finally found ourselves with complete operating systems that were free and hackers^H^H^H^H^H^H system administrators found that even with full featured operating systems such as the BSDs, there were sometimes things you would want to do that was not already in there. The way to get that something else was usually to fetch the source code, see if it would compile, make some changes (or a lot) to make it compile, possibly introduce the odd `#ifdef` block and keep at it until the software would compile, install and run. In the process you most likely found out what, if any, other software (tools or libraries) needed to be installed to complete the process. At that point, you could claim to have `/ported/` the software to your platform. If you had been careful and saved a copy of the original source files somewhere, you could use the diff utility to create a patch you could then send to the program maintainer and hope that he or she would then incorporate your changes in the next release.

But then, why wait for the next release? Why not share those diffs with others? How about putting it into a CVS repository that would be available to everyone? That idea was tossed around on relevant mailing lists for a while, and the first version of the `/ports system/` appeared in FreeBSD 1.0 in December 1993.

The other BSD systems adopted the basic idea and framework soon after, with small variations. On NetBSD, the term `'port'` was already in use for ports of the operating system itself to specific hardware platforms, so on that operating system, the ports tree is referred to as `'package source'`, or `/pkgsrc/` for short. The ports and packages tools are still actively maintained and developed on all BSDs, and most notably Marc Espie rewrote the `pkg_*` tools for OpenBSD's 3.5 release.

Parallel development has lead to some differences in the package handling on the various BSDs, and some of the operations I describe here from an OpenBSD perspective may not be identical on other operating systems. Around the same time the BSDs started including a ports tree and packages, people on the Linux side of the fence started developing package systems too. With distributed development taken to the point where the

kernel, basic system tools and libraries are maintained separately, perhaps the need there was even greater than on the BSDs. In fact, some Linux distributions such as the Debian based ones have taken the package management to the point where *everything is a package* - every component on a running system is a package that is maintained via the package system, including basic system tools, libraries and the operating system kernel. In contrast, the BSDs tend to treat the base system as a whole, with the package management tools intended solely for managing software that does not come as a part of the default install.

The anatomy of ports and packages

The ports system consists of a set of 'recipes' to build third party software to run on your system. Each port supplies its own Makefile, whatever patches are needed in order to make the software build and optionally package message files with information that will be displayed when the software has been installed.

So to build and install a piece of software using the ports system, you follow a slightly different procedure than the classical fetch - patch - compile cycle. You will need to install the ports tree, either by unpacking ports.tar.gz from your CD set or by checking out an updated version via cvs, or for that matter cvsup or the rewritten version called csup. With a populated ports tree in hand, you can go to the port's directory, say

```
$ cd /usr/ports/print/lyx
```

to see about installing lyx, the popular latex front end. On a typical OpenBSD system, that directory contains the following files:

```
$ ls -l
total 8
-rw-rw-r--  1 root  wheel  1825 May
18 21:57 Makefile
-rw-rw-r--  1 root  wheel   274 Apr
5  2007 distinfo
drwxrwxr-x  2 root  wheel   512 Nov
1  2007 patches
drwxrwxr-x  2 root  wheel   512 Nov
1  2007 pkg
```

here, the Makefile is the main player. If you open it now in a text editor or viewer such as less, you will see that the syntax is quite straightforward. What it does is mainly to define a number of variables such as the package name, where to fetch the necessary source files, which programs are required for the compile to succeed and which libraries the resulting program will need to have present in order to run correctly.

The file defines a few other variables too, and you can look up the exact meaning of each in the man pages, starting with man ports and man bsd.port.mk. With all relevant variables set, at the very end the file uses the line:

```
.include <bsd.port.mk>
```

to pull in the common infrastructure it shares with all other ports.

This is what makes the common targets work, so for example, typing:

```
$ make install SUDO=sudo
```

(probably the most common port-related make command for end users and administrators) in the port directory will start the process to install the software. But before you type that command and *press Enter*, you may want to consider this: This command will generate a lot of output, most likely more than will fit in the terminal's buffer. If the build fails, it is likely that the message about the first thing that went wrong will have scrolled off the top of your screen and out of the terminal buffer. For that reason, it is good sysadmin practice to create a record of lengthy operations such as building a port by using the script command. Typing script in a shell will give you a subshell where everything displayed on the screen will be saved in a file. Escape sequences, asterisk-style progress bars and *twirling batons* will end up a bit garbled, but that essential message you are looking for will be there too. man script will give you the details, and unless you are an incurable packrat, do remember to delete the typescript file afterwards. That process will start with checking dependencies, go on with downloading the source archive and checking that the fetched file matches the cryptographic signatures stored in the distinfo file. If the signatures match, the source code is extracted to a working directory, the patches from the patches/ directory are applied, and the compilation starts. If the dependency check finds that one or more pieces are missing, you will see that the process fetches, configures and installs the required package before continuing with the build process for the original package.

After a while, the package build most likely succeeds and the install completes. At this point you will have a new piece of software installed on your system. You should be able to run the program, and the installed package will turn up in the package listings output by `pkg_info`, such as:

```
$ pkg_info | grep lyx

lyx-1.4.3p2-qt      graphical frontend
for LaTeX (nearly WYSIWYG)
```

This information is taken from the package's subdirectory in `/var/db/pkg`, where the information about currently installed packages is stored.

If you paid close attention during the make install process, you may have noticed that the install step was performed from a binary package. This is one of the distinctive features of the OpenBSD version of the package system. The package build always generates an installable package based on a `'fake'` install to a private directory, and software is always installed on the target system from a package.

But you do not need to do that!

This means several things. If you have built and installed a package by typing `'make install'` in the relevant ports directory and later run the `'make deinstall'` or `pkg_delete` to remove the software, any subsequent install of the software will take place from the package file stored in a subdirectory of `/usr/ports/packages`. But more importantly, in most cases you can keep your system's packages up to date without a ports tree on the machine. (See Note [1]) For each release, a full set of packages is built and made available on the OpenBSD mirrors, and by the time you read this, there is reason to hope that running updates to -stable packages will be available for supported releases too.

The way to make good use of this is to set the `PKG_PATH` variable to include the packages directory for your release on one or more mirrors close to you and/or a local directory, and then run `pkg_add` with the `-u` flag. (See Note [2])

My laptop runs -current and I'm in Europe, so the PKG_PATH is set to

```
PKG_PATH=ftp://ftp.eu.openbsd.org/pub/
OpenBSD/snapshots/packages/`machine
-a`/
```

On a more conservatively run system, you may want to set it to something like

```
PKG_PATH=ftp://ftp.eu.openbsd.org/pub/
OpenBSD/4.3/packages/`machine -a`/
```

Once your PKG_PATH is set to something sensible, you can use pkg_add and the package base name to install packages, so a simple

```
$ sudo pkg_add lyx
```

would achieve the same thing as the `'make install'` command earlier, and most likely a lot faster too. Once you have a set of packages installed, and keeping in mind that you need a meaningful `PKG_PATH`, you can keep them up to date using `pkg_add -u`. If you want more detailed information about the package update process and want `pkg_add` to switch to interactive mode when necessary, you can use something like this command:

```
$ sudo pkg_add -vui
```

I have at times tended to run my pkg_add -u with some of the -F flags in order to force resolution of certain types of conflict, but given the quality of the work that goes into the packages, most of the `-F` options are rarely needed.

`pkg_add` and its siblings in the `pkg_*` tools collection has a number of options we have not covered here, all intended to make your package management on OpenBSD as comfortable and flexible as possible. The tools come with readable man pages, and may very well be the topic of future BSD Magazine articles.

More information on the net

The main source of information about the OpenBSD ports and packages system is to be found on the OpenBSD project's web site. The FAQ's ports and packages section at *http://www.openbsd.org/faq/faq15.html* has more information about all the issues covered in this article, and goes into somewhat more detail than space allows here. If you encounter problems while installing or managing your packages, it is more than likely that you will find a solution or a good explanation there. And of course, if nothing else works or you can't figure it out, there is always the option of asking the good people at *misc@openbsd.org* or *ports@openbsd.org* or search the corresponding mailing list archives.

How do I make a package then?

That is a large question, and the first question you should ask if you think you want to port a particular piece of software is, *Has this already been ported?*. There are several ways to check. If you are thinking of creating a port, you most likely already have the ports tree installed, so using the ports infrastructure's search infrastructure is the obvious first step. Simply go to the `/usr/ports` directory and run the command:

```
$ make search key=mykeyword
```

Where mykeyword is a program name or keyword related to the software you are looking for. One other option with even more flexible search possibilities is to install databases/sqlports. And of course, searching the ports mailing list archives (*http://marc.info/?l=openbsd-ports*) or asking the mailing list works too.

When you have determined that the software you want to port is not already available as a package, you can go on to prepare for the porting effort. Porting and package making is the subject of much usenet folklore and rumor, but in addition you have several man pages with specific information on how to proceed. These are, ports, package, packages, packages-specs, library-specs and `bsd.port.mk`.

Read those and use your familiarity with the code you are about to port to find your way. The OpenBSD web offers a quite a bit of information too. You could start with re-reading the main ports and packages page at *http://www.openbsd.org/faq/faq15.html*, and follow up with the pages about the porting process at *http://www.openbsd.org/porting.html*, testing the port at *http://www.openbsd.org/porttest.html* and finally the checklist for a sound port at *http://www.openbsd.org/checklist.html*.

All the while, try first to figure out the solution to any problems that pop up, read the supplied documentation, and only then ask port maintainers via the ports mailing list for help. Port maintainers are generally quite busy, but if you show signs of having done your homework first, there is no better resource available for helping you succeed in your porting or port maintenance efforts.

One fine resource for the aspiring porter is Bernd Ahlers' ports tutorial from OpenCon 2007, you can look up Bernd's slides at *http://www.openbsd.org/papers/opencon07-portstutorial/index.html*, and it is possible he can be persuaded to repeat the tutorial at a conference near you.

Notes

[1] The main exceptions to the rule that precompiled packages are available from the mirrors are software with licenses that do not allow redistribution or require the end user to do specific things such as go to a web site and click a specific button to formally accept a set of conditions. In those cases it cant' be helped, and you will need to go via the ports system to create a package locally and install that.

[2] If you want to find out what packages are available at your favorite mirror, you can get a listing of package names by fetching the file `$PKG_PATH/index.txt`. The OpenBSD web site offers a listing of available packages with short descriptions, too. For OpenBSD 4.3, the listing is available from *http://www.openbsd.org/4.3_packages/*, from there you click on the link for your platform

About the Author

Peter N. M. Hansteen is the author of The Book of PF (No Starch Press, December 2007). Peter has been tinkering with computers and networks since the mid-1980s, found the Freenixes in the early 1990s and is a frequent lecturer on PF and other OpenBSD and FreeBSD topics. He is a consultant, sysadmin and writer based in Bergen, Norway who occasionally blogs at http://bsdly.blogspot.com/ and welcomes your comments to peter@bsdly.net.