HOW TO Build a Port

by Erwin Lansing / FreeBSD has long been known for its ports tree, where users had to start by compiling all the software they wanted to install, which in some cases can be slow and bothersome. Over the last few years, a lot of work has been done to build a first-class packaging system, where a user can simply download and install a pre-built package, with entirely new packaging tools and also a brand new package distribution infrastructure. For most users, this is now the recommended way to install software, and most users will no longer need to compile software themselves if using the new pkg(8) system (see *FreeBSD Journal*, March/April 2014). Underneath it all, however, is still the FreeBSD ports system, which has all the information on how to build and install third-party software on FreeBSD. With almost 25,000 packages available, it still may happen that a package does not exist, and so a new port will need to be written. Similarly, to change an existing package, the underlying ports need to be modified. In this article I give a brief overview of how the ports system is structured and an introduction on how to build a simple port.

First you need to get a fresh copy of the FreeBSD ports tree. The fastest way to get one is by using portsnap(8). This is a two-step process. First fetch the portsnap data file, which is about 70Mb and can take some time depending on your Internet connection. Then install the ports tree itself, which will extract a large number of small files, which also can take time depending on the speed of your disks.

```
portsnap fetch
portsnap extract
```

This will place a full copy of the ports tree in /usr/ports. Updating an earlier version of the ports tree installed by portsnap is a similar two-step process:

```
portsnap fetch
portsnap update
```

Looking in /usr/ports you can see a number of files and a large number of directories. One of the first files that should be noticeable is the Makefile. The FreeBSD ports tree is written in make(1), a language extensively used for building and installing software. Of course a single 200 line Makefile is not enough, and most of the ports infrastructure code is in the /usr/ports/Mk directory. Most of the other directories are the categories containing the actual ports.

A very simple port consists of four files: Makefile, pkg-descr, pkg-plist, and distinfo. The Makefile contains some basic information about the port and the instructions on how to fetch, build, and install it. pkg-descr, or package description, contains a short description of what this port is and does and also a link to the official project website of the software package. pkg-plist, or packaging list, is a list of files that this port installs, and finally, distinfo contains a checksum and size of any external files this port needs to fetch from the Internet. I'm going to use the Stand-Alone Shell (SASH) port, found in /usr/ports/shells/sash, as an example here. In Example 1 you can also see a directory named files, which can contain additional files useful for this port.

Example 1:

```
erwin@panda:/usr/ports/shells/sash % ls -l
total 20
-rw-r--r--  1 root  wheel  349 Apr 11  2014 Makefile
-rw-r--r--  1 root  wheel  123 Apr 11  2014 distinfo
drwxr-xr-x  2 root  wheel  512 Jun 24  2014 files
-rw-r--r--  1 root  wheel  458 Jan 22  2014 pkg-descr
-rw-r--r--  1 root  wheel   35 Jun 11  2014 pkg-plist
```

Example 2:

```
erwin@panda:/usr/ports/shells/sash % cat Makefile
# Created by: Patrick Gardella <patrick@FreeBSD.org>
# $FreeBSD: head/shells/sash/Makefile 350947 2014-04-11 13:41:06Z miwi $

PORTNAME=          sash
PORTVERSION=       3.8
CATEGORIES=shells
MASTER_SITES=      http://members.tip.net.au/~dbell/programs/

MAINTAINER=        ports@FreeBSD.org
COMMENT=           Stand-Alone Shell combining many common utilities

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

Looking at the SASH Makefile in Example 2, the first line has an attribution to the original author when the port first was added to the ports tree. Next is the subversion identification string, which will be expanded by subversion automatically and for a new port should only be:

```
# $FreeBSD$
```

Next is a section of variables with some essential information about the software this port will install. The name of the port in PORTNAME should be unique across the whole ports tree, although there are options available for adding a prefix or suffix to the name.

PORTVERSION is the version of this port and is usually the same as the version of the original software package. Note that the ports version can never decrease, as the tools for upgrading installed packages cannot handle versions going back in time.

CATEGORIES is a list of port categories starting with the primary category this port belongs to, which is also the directory in the ports tree it is in. A port can belong to more than one category listed after the primary category including virtual categories that do not have a directory themselves, but can still help users find the port. A full list of available categories is available in the Porters Handbook: https://www.freebsd.org/doc/en_US.ISO8859-1/books/porters-handbook/makefile-categories.html#porting-categories.

The list of URLs in MASTER_SITES is used by the ports system to fetch the original sources of the software package. It is recommended to have more than one in case that site is temporarily unreachable. This is also where some of the magic starts. The actual URL of the file to be fetched is constructed by the ports system automatically, based on some of the earlier specified variables. In its simplest form it will default to

```
${MASTER_SITE}/${PORTNAME}-${PORTVERSION}.tar.gz
```

In the example SASH port, this will be:

```
http://members.tip.net.au/~dbell/programs/sash-3.8.tar.gz
```

Lots of software projects use their own versioning or naming schemes, and there are a large number of additional variables available to influence the final URL, e.g., setting USE_BZIP2 will change the default suffix to .tar.bz2, and USE_XZ will change it to .tar.xz. There are also macros available for well-known download sites with a large number of mirrors, like SourceForge and CPAN. Another interesting and extensively used site is github, which does not have a concept of official released files, and so the port will need to depend on a given commit and tag to ensure it downloads a consistent and tested version of the software. These are all described in the Porters Handbook: https://www.freebsd.org/doc/en_US.ISO8859-1/books/porters-handbook/makefile-distfiles.html.

The next section contains some basic metadata about the port. The FreeBSD ports system has a concept of maintainership, where a single person or multiple people behind a mailing list address are the primary contact for the port. Requesting changes to a port, like updating to a newer version, should be approved by the maintainer, and filing a bug report in the FreeBSD bugzilla instance will automatically generate an email to the address listed in the MAINTAINER field.
COMMENT is, as you may have guessed, a short, one-sentence description of the port. A longer description is available in the pkg-descr file.

Finally, the port needs to invoke the ports infrastructure itself by including the main file:

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

Example 3:

```
erwin@panda:/usr/ports/shells/sash % cat pkg-descr
SASH (Stand-Alone Shell)

It is a nice combination of bare-bones shell and a dozen
or so most useful UNIX commands.

Shell includes: echo pwd cd mkdir mknod rmdir sync rm chmod
                chown chgrp touch mv ln cp cmp more exit
                setenv printenv umask kill where

Commands include: dd ed grep gzip ls tar file find mount chattr
WWW: http://members.tip.net.au/~dbell/
```

Speaking of the pkg-descr file, Example 3 lists the contents of pkg-descr for the shells/sash port, which is quite explicit and detailed. A usual pkg-descr should only have a few paragraphs concisely describing the port so a user knows what the port does without having to read documentation or visit a website. The official website of the software project can be included on the last line, prepended by WWW.

Example 4:

```
erwin@panda:/usr/ports/shells/sash % cat pkg-plist
@shell bin/sash
man/man1/sash.1.gz
```

The files installed by the port are listed in the pkg-plist file. This list is used at several stages in a port's lifetime. The tools that generate packages use this list to know what files to include, and a tool removing an installed port or package also needs to know which files it should delete. Files are installed in ${PREFIX}, which defaults to /usr/local, is implicitly assumed in the pkg-plist file, and should not be included. As you can see in Example 4, our small port only installs one executable file and its corresponding man-page. More strengths of the ports system turn up here. Keywords can be used in the packaging list to handle more advanced cases where just installing a given file in a given location needs more care, or where an additional file needs to be modified. The @shell keyword in our example not only installs the sash executable in /usr/local/bin, but also adds it to /etc/shells so that it becomes a recognized shell on the system. Another example is the @sample keyword. This is very useful for installing configuration files that should not be overwritten by the stock version in a later upgrade. Using the @sample keyword, the port or package will install the stock version as filename.conf.sample, but before that it will test whether filename.conf exists, and if so, whether it is the same as the stock file in the previous version, and if not, or if it doesn't exist, install the stock version as filename.conf as well. In other words, if the configuration file already exists and was modified by the user since its installation, those modifications will not be overwritten.
More keywords can be found in the /usr/ports/Keywords directory.

Finally we get to the distinfo file. This is also the easiest file to create. If all the variables in the Makefile were correctly filled in, especially those that influence how to fetch the original sources, it is just a matter of running:

```
make makesum
```

This will fetch the files needed and generate the distinfo file with a SHA256 checksum and the size of the file. See Example 5.

The files directory in a port can hold any file the port needs at some point during the build, installation, or packaging. Filenames starting with the prefix patch- will automatically be applied between unpacking the original sources and compilation and can be very useful when porting software written without FreeBSD in mind. See Example 6.

For a new port, an entry in the category Makefile needs to be created. In our example, that would be in /usr/ports/shells/Makefile

```
SUBDIR += sash
```

Dependencies

One important and often used feature our example port does not use is dependencies. A dependency is another port that the current port needs in some phase of its lifetime, for example, an executable or other file during run time, or while building, or a library. Libraries are specified in the LIB_DEPENDS variable as a list of library:port combinations.

Example 5:

```
erwin@panda:/usr/ports/shells/sash % cat distinfo
SHA256 (sash-3.8.tar.gz) =
13c4f9a911526949096bf543c21a41149e6b037061193b15ba6b707eea7b6579
SIZE (sash-3.8.tar.gz) = 53049
```

Example 6:

```
erwin@panda:/usr/ports/shells/sash % ls -l files/
total 20
-rw-r--r--  1 root  wheel  1356 Apr 11  2014 patch-Makefile
-rw-r--r--  1 root  wheel   442 Jan 22  2014 patch-cmd_ls.c
-rw-r--r--  1 root  wheel  2572 Apr 11  2014 patch-cmds.c
-rw-r--r--  1 root  wheel   551 Apr 11  2014 patch-sash.c
-rw-r--r--  1 root  wheel   219 Jan 22  2014 patch-sash.h
```

Example 7:

```
erwin@panda:/usr/ports/shells/sash % make describe
sash-3.8|/usr/ports/shells/sash|/usr/local|Stand-Alone shell combining many common
utilities|/usr/ports/shells/sash/pkg-descr|ports@FreeBSD.org|shells||||||http://members.tip.net.au/~dbell/
```

```
        LIB_DEPENDS=
libsqlite3.so:${PORTSDIR}/databases/sqlite3
```

Run time or build time dependencies are specified in a similar way in the RUN_DEPENDS and BUILD_DEPENDS variables, consisting of a list of file:port combinations.

```
        RUN_DEPENDS=
${LOCALBASE}/bin/bash:${PORTSDIR}/shells/bash
```

The ${LOCALBASE} variable is usually the same as ${PREFIX} we saw earlier, but make the distinction between where already installed ports are (LOCALBASE) and where the current port will be installed (PREFIX). For an executable in the default path, the full path is not needed, so the above example could be shortened to:

```
        RUN_DEPENDS=
bash:${PORTSDIR}/shells/bash
```

Testing the Port

It is highly recommended to turn on the developer mode while testing. This is easily done by setting DEVELOPER=yes in /etc/make.conf:

```
        # echo DEVELOPER=yes >> /etc/make.conf
```

This turns on extra quality checks and displays more warnings, like use of deprecated features.

A first and simple test that can be done is running "make describe." This should produce a string composed of a lot of the metadata we talked about earlier. This information is used by several tools for keeping track of dependencies and searching for port names or descriptions. See Example 7.

The portlint utility, found in ports-mgmt/portlint or just "pkg install portlint," evaluates whether the port is syntactically correct and also has some recommendations for best practices that have developed over the years. For a new port, portlint -A should be run; for an existing port portlint -C is sufficient. See Example 8.

Example 8:

```
erwin@panda:/usr/ports/shells/sash % portlint -C
WARN: Makefile: Consider defining LICENSE.
0 fatal errors and 1 warning found.
```

Example 9:

```
LICENSE=        GPLv2
LICENSE_FILE=   ${WRKSRC}/COPYING
```

This is a valid warning as our example port does not set the ${LICENSE} variable. The ${LICENSE} variable can be set to a known license abbreviation, listed in /usr/ports/Mk/bsd.licenses.db.mk. If a software project includes a file with its license, it should also be installed by using the ${LICENSE_FILE} variable. See Example 9.

Testing a Port

The easiest way to test is, of course, to install the newly hatched port on your local system. I would recommend against that, as it can easily break the existing system, accidentally overwriting an existing file or exhibit other unexpected behavior, and already installed software can also make it harder to check what the new port adds. For example, with plenty of software already installed, it is easy to miss a newly installed file and leave it out of the pkg-plist.

Using a clean installation or a jail might help avoid most of those pitfalls, but tools are available to do all the hard work for you. The poudriere system (https://github.com/freebsd/poudriere) is not only useful for building your own in-house package repository, but is also an excellent port testing framework. It can be used to test the entire ports tree, a subset of the tree, or even a single port and its dependencies. Using jails and the ZFS file system, it ensures that it is self-contained and does not influence the host system in any way. It generates very useful log files on a per-port basis, that can be used to see how the port was built, packages installed and deinstalled, and is very useful for checking whether the pkg-plist file was complete. It can be a bit hardware hungry due to its heavy reliance on ZFS, and of course compiling does require a lot of processing.

Install poudriere from a package "pkg install poudriere" or from the port in ports-mgmt/poudriere, and review the settings in /usr/local/etc/poudriere.conf, especially those dealing with the ZFS file system. Next, create a jail with the FreeBSD version you want to test:

```
# poudriere jail -c -j 93Ramd64 -v 9.3-RELEASE -a amd64
```

If you plan to submit your port to be included in the FreeBSD ports tree, you should test on all supported major releases. The ports team supports the same FreeBSD releases as the FreeBSD security officer (https://www.freebsd.org/security/), which currently includes FreeBSD 8, 9, and 10, plus the development version in HEAD, 11.

Of course you need a ports tree as well, which can be created by:

```
# poudriere ports -c
```

This will create a ports tree named default. You're now ready to test your port:

```
# poudriere testport -j 93Ramd64 -p default -o shells/sash
```

You will find the log file in /poudriere/data/logs/bulk/93Ramd64-default/latest/logs/sash-3.8.log. A more extensive introduction for using poudriere as a testing tool can be found in the Porters Handbook: https://www.freebsd.org/doc/en/books/porters-handbook/testing-poudriere.html.
At the time this article was written, the redports (https://redports.org) system was down for maintenance. Hopefully, the site will be back, so keep checking. It provides all the tests above and a large number of extended checks, all via a convenient web interface.

Submitting a Port

Once you are satisfied the port works the way you want, you are ready to submit it for inclusion in the official FreeBSD ports tree, which will also make packages available for easy installation with pkg. Make sure the port directory is clean of any unnecessary files. Removing the work/ directory can easily be done with the "make clean" command. From the category directory, one level up from the port directory, use the shar(1) utility to bundle the files in a shar archive:

```
shar `find sash'> sash.shar
```

The resulting sash.shar can be submitted to the FreeBSD bugzilla database.

We have only brushed the surface in this introduction to the ports tree, so be sure to check the Porters Handbook (https://www.freebsd.org/doc/en/books/porters-handbook/index.html), which is constantly updated and contains comprehensive information on all the features of the ports tree. There is also an active porters community that will be happy to answer any questions. You can find them on the freebsd-ports mailing list (http://lists.freebsd.org/mailman/listinfo/freebsd-ports) or on the IRC channel #bsdports on EFnet. Happy porting! ●

Erwin Lansing lives in Copenhagen with his wife and son. He works for DK Hostmaster trying to keep the lights on on the Danish Internet. He is also Vice President of the Board