The FreeBSD Ports Collection

Satoshi Asami, The FreeBSD Project
asami@freebsd.org
http://www.freebsd.org/ports/

Overview

FreeBSD is an open source operating system based on 4.4BSD-Lite2, a version of UNIX from the University of California at Berkeley. It is maintained by a group of volunteers from around the world. In addition to providing a complete operating system, the FreeBSD project supports an extensive collection of sanctioned third-party software called *the Ports Collection*, many of which were contributed by the users. In addition to the source form, most of the ports are provided as binary packages too.

Port Skeletons and Distfiles

The Ports Collection consists of a single `make` macro file, `bsd.port.mk`, and some skeleton files for each port that describe how to compile and install the software. If there were changes made to the original software to compile it on FreeBSD, patches to reproduce those changes are included too.

One of the items specified in the port's `Makefile` is the name and URL where the source files ("*distfiles*") of the original software are located. When the user attempts to compile a port, they are fetched over the Internet if they do not exist on the system. The distfiles are checksum-verified to ensure consistency, as well as guarding against possible Trojan horse attacks.

Since the distfiles can be fetched on demand, this allows the Ports Collection itself to stay small. For instance, with over 2,200 ports as of April 1999, all the skeleton files total only about 70MB. In contrast, the entire set of distfiles, most of which are compressed archive files, are over 1.4GB. It takes over 8GB to extract and compile them all at once.

Packages

In addition to providing an easy way to compile programs, `bsd.port.mk` provides a set of commands to create binary *packages* of installed ports. These packages, which are compressed archive files with some additional information, can be installed on a FreeBSD system using the `pkg_add` command. They contain a listing of the entire set of installed files, so they can also be deleted completely, using the `pkg_delete` command. Each release of FreeBSD ships with a complete set of packages. Currently, there are about 1.2GB of packages.

The Ports Collection framework always supported a simple top-down build of packages. In other words, when the user types "`make package`" at the root directory of the ports hierarchy, `bsd.port.mk` will arrange for the build process to go into every single subdirectory and build packages for each of them, one by one. This method, akin to the way many large software trees are built, has exhibited many problems as the Ports Collection grew.

Dependencies

Many software depend on others to build or run. These are called *dependencies*. If port A requires port B, then port A is called the *dependent* port and port B is the *dependency*. The Ports Collection has a mechanism of handling dependencies automatically. When a dependent port is built, the environment is checked to see if the dependency is already installed, and if not, the dependency is built and installed first. The dependency check is recursive, so if a dependency requires another port, it will also be checked and built. This chain can continue to an arbitrary depth.

The dependency information is also recorded in packages. When a package is installed, `pkg_add` checks if all the dependencies are installed as well, and if not, they will be installed automatically. Again, package dependency checking is recursive, and installing one package could potentially pull in dozens of other packages.

One of the interesting applications of the dependency mechanism is to create *meta-packages*, i.e., empty packages that make it easy for users to install several packages at once.

Problems

There were many issues that that had to be addressed as the Ports Collection grew from less than 200 ports in January 1995 to over 2,200 ports in April 1999. In addition to providing a brief summary of the history of the Ports Collection, this talk addresses the problems, such as shared library conflicts, dependency detection, etc., and how we resolved them. I will also describe the process we use to build the 2,000 packages in a few hours before releases.