Tracking FreeBSD in a Commercial Setting

M. Warner Losh
Cisco Systems
Broomfield, CO
imp@freebsd.org

Abstract

The FreeBSD project publishes two lines of source code: current and stable. All changes must first be committed to current and then are merged into stable. Commercial organizations wishing to use FreeBSD in their products must be aware of this policy. Four different strategies have developed for tracking FreeBSD over time. A company can choose to run only unmodified release versions of FreeBSD. A company may choose to import FreeBSD's sources once and then never merge newer versions. A company can choose to import each new stable branch as it is created, adding its own changes to that branch, as well as integrating new versions from FreeBSD from time to time. A company can track FreeBSD's current branch, adding to it their changes as well as newer FreeBSD changes. Which method a company chooses depends on the needs of the company. These methods are explored in detail, and their advantages and disadvantages are discussed. Tracking FreeBSD's ports and packages is not discussed.

1 Problem Statement

Companies building products based upon FreeBSD have many choices in how to use the projects sources and binaries. The choices range from using unmodified binaries from FreeBSD's releases, to tracking modify FreeBSD heavily and tracking FreeBSD's evolution in a merged tree. Some companies may only need to maintain a stable version of FreeBSD with more bug fixes or customizations than the FreeBSD project wishes to place in that branch. Some companies also wish to contribute some subset of their changes back to the FreeBSD project.

FreeBSD provides an excellent base technology with which to base products. It is a proven leader in performance, reliability and scalability. The technology also offers a very business friendly license that allows companies to pick and choose which changes they wish to contribute to the community rather than forcing all changes to be contributed back, or attaching other undesirable license conditions to the code.

However, the FreeBSD project does not focus on integration of its technology into customized commercial products. Instead, the project focuses on producing a good, reliable, fast and scalable operating system and associated packages. The project maintains two lines of development. A current branch, where the main development of the project takes place, and a stable branch which is managed for stability and reliability. While the project maintains documentation on the system, including its development model, relatively little guidance has been given to companies in how to integrate FreeBSD into their products with a minimum of trouble.

Developing a sensible strategy to deal with both these portions of FreeBSD requires careful planning and analysis. FreeBSD's lack of guidelines to companies leaves it up to them to develop a strategy. FreeBSD's development model differs from some of the other Free and Open Source projects. People familiar with those systems often discover that methods that were well suited to them may not work as well with FreeBSD's development model. These two issues cause many companies to make poor decisions without understanding the problems that lie in their future.

Very little formal guidance exists for companies wishing to integrate FreeBSD into their products. Some email threads can be located via a Google search that could help companies, but many of them are full of contradictory information, and it is very disorganized. While the information about the FreeBSD development process is in the FreeBSD handbook, the implications of that process for companies integrating FreeBSD into their products are not discussed.

2 FreeBSD Branching

The FreeBSD development model strikes a balance between the needs of the developers and the needs of its users. Developers prefer to have one set of sources that they can change arbitrarily and not have to worry about the consequences. Users prefer to have a stable system that is compatible with the prior systems. These two desires are incompatible and can cause friction between developers and users.

FreeBSD answers the need of both groups by providing two versions of its code. The project maintains a main line for its developers, called “current.” This branch contains all the latest code, but all that code might not be ready for end users. All changes to FreeBSD are required to be first committed to the current branch. The quality of the current branch varies from extremely stable to almost unusable over time. The developers try to keep it towards the stable end of the spectrum, but mistakes happen.

To provide a stable system users can use, FreeBSD also maintains a stable version of the OS. Every few years the current version is branched and that branch becomes the new stable version. This branch is called either “stable” or “RELENG X” where X is the major version number for that branch. Stable branches are well tested before they are released. Once released, only well tested patches from the current branch are allowed to be merged into the branch. Once a stable branch is created, its ABI and API are never changed in an incompatible manner, which allows users to upgrade to newer releases that are made from the stable branch with relative ease. Stable branches tend to have a lifetime of about 2-6 years.

An even more stable version of FreeBSD is available than the stable branch. For each release made off a stable branch, a release branch is also created. The only changes that go into these release branches are security fixes and extremely important bug fixes. These are designed for users that wish to run a specific release, but still have high priority bugs fixed and available in a timely fashion. Since release branches are targeted only at end users and have so few changes, the rest of this paper will treat them as a release.

Figure 3 tries to show the relationships between the different branches over time. It shows what should have theoretically happened if FreeBSD had a major release every two years. The horizontal axis is time (in years). The vertical axis is the amount of change, in arbitrary units. Vertical arrows point to the theoretical release points (with the release number under the arrow when the name fits). After about three years, the branches stop being used in favor of newer releases.

Figure 4 shows data from the FreeBSD project since early 1997.[1] There are many features of this graph that differ from the idealized graph. The two that are most relevant are that major branches live beyond the three year idealized vision and that the timing of the release branches isn’t completely regular. These points will be important later in deciding which method fits the company’s needs the best.

The FreeBSD ports system (which is used to generate the packages that appear in FreeBSD’s releases) is not branched at all. Instead, it supports both the current branch, as well as the active stable branches of the project. For each release, the tree is tagged so that it can be reproduced in the future if necessary. These policies are different than the main source tree. Tracking of the ports tree is not addressed further in this paper because its model is different and the author has fewer examples from which to draw advise and conclusions from.

3 Branching Choices

There are a wide range of companies using FreeBSD in their products today. On the simplest end, companies load FreeBSD onto boxes that they ship. On the most complex end, companies modify FreeBSD extensively to make it fit their needs. Over the years four different approaches to tracking FreeBSD have evolved.

The simplest method involves using the stock FreeBSD releases unmodified. Companies doing this grab FreeBSD at its release points and make no changes to the software and just configure th system and install the packages that their customers need. Typically no sources are tracked and only binary packages from FreeBSD’s web pages are used.

The next simplest method involves grabbing a release of FreeBSD and using that as a basis for their product. FreeBSD is effectively forked at this point as the company makes whatever modifications are necessary for their product. No thought is given to upgrades or contributing bug fixes back into the community.

[1] The author used fairly simple scripts to extract this data from the commit logs, whose format changed in 1997. Some flaws exist in the data, but they do not affect the shape of the graph.

Companies often setup repositories of FreeBSD stable branches. In this model, the tip of a stable branch (or the latest release point) is imported into some SCM. The company will then make fixes and improvements to its private branch. The company will import newer versions of FreeBSD on this stable branch from time to time. Better run companies will try to contribute their fixes back into FreeBSD to simplify their upgrade path.

The most complicated method involves mirroring the FreeBSD development process. The company will import the latest version of the FreeBSD development branch. They will setup automated scripts for pulling in newer versions. They will make their changes to FreeBSD in this mainline of development. Rather than using FreeBSD's stable branches, the company will decide when and where to branch its version. Once branched, it will control what fixes are merged into its branch.

3.1 Stock FreeBSD

The most widespread use of FreeBSD involves this method. In this method, the company grabs the binaries from a FreeBSD web site or commercial vendor and uses them as built. They layer packages on top of FreeBSD, typically a mix of stock packages from the release and their own additional scripts or programs. The focus of these companies is to have a system that they can deploy and use for a particular purpose.

Customization of the system is typically tracked in some kind of source code management (SCM) system. These customizations include the `/etc/rc.conf` file (which controls most of the global settings for the system), as well as configuration files and other data used by the system. Some of these companies will also compile customized kernel configurations. These files can typically be tracked in any SCM as the demands on the SCM are modest.

These companies typically upgrade only when they need to do so. Once they find a stable version they stick with it until they need something from a newer version. This could be support for newer hardware (drivers or architectures), or application level features such as threading support. Often times they will track newer security releases with services such as FreeBSD update and/or portupgrade in package mode.

FreeBSD meets the needs of these companies fairly well. They don't require additional features or bug fixes not in the current releases. They don't need to optimize FreeBSD for any given platform beyond what the standard system tunables provide for them. The main advantage for these companies is that FreeBSD is a drop in solution. There's very little overhead necessary to get their machines and applications running and FreeBSD's standard install tools can be used to create images for their products (if they even need separate images at all). Some of these companies participate in the community and contribute to the community in many ways. Some of these companies do not. The choice is up to the individual company and its needs, sensitivities and desires.

3.2 Grab and Go

Another easy way to use FreeBSD sources is the grab and go method. In this method, the companies grab FreeBSD at some version and then never upgrade FreeBSD. No attempts to track FreeBSD or pull bug fixes in from FreeBSD are made. The company grabs the source and starts hacking. They layer in their own build and packaging system often times. Sometimes they port to a new architecture. FreeBSD typically is the base for a more extensive application of appliance which the company has total control over.

There are a few advantages to this method. The company can concentrate on making their product work without the distractions introduced when software versions are rolled. The company manages its risk by doing everything themselves. The company can keep any information about what they are doing from being inferred by competitors looking at their bug submissions to FreeBSD. The company's employees are not distracted by interactions with the FreeBSD community. Without these distractions, it is believed that this method allows a company to bring its product to market more quickly.

However, there are many disadvantages to this method. The biggest problem is that companies using this method often find it difficult to get support for the community. Most of the active members in the community have moved on to newer versions of the software, so are unable to help out with problems in older versions. Many of the bug fixes in newer versions of the software are difficult to back port because they depend on other changes to the software that aren't present in the older versions of the software. Often times, interaction with the community on problems for recent releases of the software can save tremendous amounts of time for the company's employees because they can leverage the knowledge of others who have had similar problems.

Companies often times think they are in total control of the hardware platform, but in reality this is a mistaken assumption. Hardware platforms are made of up chips that one buys from manufacturers. These chips go obsolete at an alarming rate sometimes, forcing changes to the underlying hardware to even be able to continue to build it. These new chips often times require new changes to the software. Just as often, others in the community have used the newer parts and have migrated the necessary changes into FreeBSD. So often times companies that go down this path are forced to redo work that has already been done in the community when their supplies tell them that they will no longer be able to give them a certain chip, and no replacements from other vendors exist.

Some companies have managed to start out with this method and later transition to one of the other methods described in this paper. One is even rumored to have recently completed the jump from FreeBSD 2.1.6 (released in 1996) to FreeBSD 6.2 and are now using the stable branch tracking method described below. Other times, the outcome isn't so good and the product is migrated to another system, or the product is killed.

3.3 Stable Branch Tracking

One nice feature of FreeBSD's stable branches is their stability. One can typically count on them to build and not have critical problems. The stable branch tracking strategy takes advantage of this feature.

The first major release on a branch is imported into a private SCM for the company to use. The sources are imported using the 'vendor branch' facility of the SCM. This facility allows one to keep a pristine copy of the sources from FreeBSD separate from the modified sources for the company. This separation allows developers to produce patches between the two. These patches can be used to determine which changes should be contributed back to the FreeBSD tree. In addition, by importing into a vendor branch and merging into the company's private branch, the company can upgrade versions of FreeBSD at any time. They can pull either a whole new FreeBSD tree, or individual files that have the fixes they need. The company can choose when to roll forward the basis of their tree, and the branching features of most SCMs make this procedure relatively easy. As new stable branches of FreeBSD become available, this process can be repeated for them in a separate module or directory in the SCM.

The big advantage to this approach is the underlying nature of the stable branch itself. The FreeBSD project has policies and practices that ensure that the branch will be stable, especially near releases off of that branch. The ability to "cherry pick" fixes from newer versions of FreeBSD without affecting the rest of the branch helps to mitigate risks associated with upgrading. In addition, by using the vendor branch feature, these changes will not interfere with future imports of a more complete system when it is appropriate to do so. Since the ABI and API are also frozen for the entire branch, one can grab fixes and changes from newer versions without worrying about breaking applications under development within the company. The isolation of major releases into separate modules in the SCM allows a company that has several products built on FreeBSD to selectively upgrade them to newer versions as market conditions warrant.

There are a few disadvantages for this approach. First, to fully leverage the FreeBSD community, it is desirable to push back bug fixes to the community in a timely fashion. When this isn't done, as is often the case when deadlines are tight, the chore up upgrading increases because one must bring forward all of the changes to the system. Second, if the company makes extensive changes that aren't merged back into FreeBSD and want to migrate to the next major version, they will need to redo their changes after the next major branch is created. If they are in an area of FreeBSD that has changed between the two branches, this can take quite a bit of time and effort.

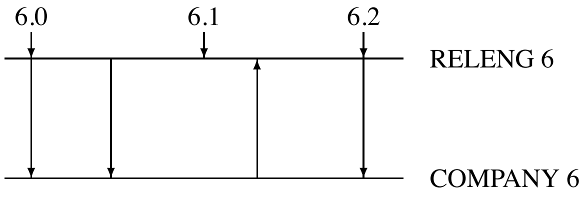

Figure 1: Code Flow between FreeBSD RELENG 6 and Company's Version

Figure 1 shows this graphically. This figure shows an idealized flow of patches into the company tree and back to FreeBSD. It also neglects to picture the required trip through FreeBSD current required for all patches to be committed to stable branches. The number of changes to the branches are also abstracted out, unlike Figure 1 and 2 presented above. The arrows pointing to the RELENG branch represent FreeBSD releases from that branch. The arrows from the RELENG branch to the COMPANY branch represent merges of code from FreeBSD into the company's repository. The arrows from COMPANY to RELENG represent patches that have successfully been contributed back into FreeBSD and have been merged into FreeBSD's RELENG tree.

3.4 Own Branching

One way to keep current in FreeBSD is to track FreeBSD's main development branched called "current." Many developers do this in the FreeBSD perforce tree and it works well for them. This method follows that practice, but also adds stable branches, akin to FreeBSD's stable branches in concept, but not tracking any specific FreeBSD release.

The company would import FreeBSD's current code as its starting point for its FreeBSD development efforts. They would start making changes to their current branch. In addition, source code pulls from FreeBSD's current branch would be frequent to keep the company's current branch close to FreeBSD's current branch. Just after these pulls, the company's current branch would be exactly FreeBSD's current branch with only the company's changes layered on. The company would then merge the relevant change from its current tree into FreeBSD's current tree by working with the FreeBSD community to produce acceptable patches.

The company would also emulate FreeBSD's branching practices. When the tree is in a good state to branch, possibly driven by delivery schedules for its end products, the company would branch its own stable branch from their current branch. They would merge bug fixes and new features from their current branch into this stable branch and build products from this stable branch.

The main advantage of this approach is that it is easier to keep current with FreeBSD than the stable branch tracking approach. To generate patches, a simple diff(3) between the FreeBSD sources and the company sources will generate the patches. As patches are merged with FreeBSD, the next pull will automatically include those changes and the delta between the company's sources and FreeBSD's will drop. By controlling the branching times, there's no need to wait for FreeBSD to create new a stable branch, so the company can drive released schedules more easily than companies tracking stable branches.

The main disadvantage of this approach is that the company loses the work done by the FreeBSD community to keep its stable branches stable and useful. Since there is no connection between the company's stable tree and FreeBSD's stable tree, improvements to FreeBSD's stable branch aren't automatically reflected in the company's stable branch. An engineer will need to watch changes going into either the current branch from FreeBSD, or into FreeBSD's stable tree and manually pull them into their own stable branch. Typically, there are on the order of 100-200 commits to a FreeBSD stable branch a month, so this load can be quite large. In addition, except around the time a new branch is cut in FreeBSD, FreeBSD's current branch may have periods of instability and it can be quite difficult to know when a good time to branch might be as many of the stability or quality problems that are in FreeBSD's current branch often lay undiscovered for months or years because it doesn't get the intensity of testing that a FreeBSD stable branch receives.

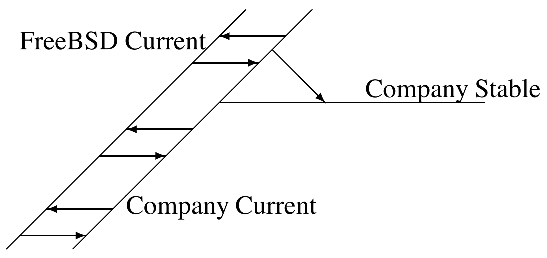

Figure 2: Relationship between FreeBSD current and company branches

Figure 2 shows this graphically. This figure shows an idealized flow of patches into the company tree and back to FreeBSD. The two parallel current branches are shown diagonally, with the company's custom stable branch shown horizontally, much like Figures 1 and 2 presented above. No FreeBSD release points are included, since they are largely irrelevant to the method. The exact delta between the two current branches is also abstracted out, as this will ebb and flow over time and needlessly complicates the graph. The arrows represent changes being merged from one branch to another, either between the two current branches, or from the company's current branch to its stable branch.

4 Acknowledgments

I would like to thank the crew at Timing Solutions: Ben Mesander, John Hein, Patrick Schweiger, Steve Passe, Marc Butler, Matthew Phillips, and Barb Dean for their insight and implementation of the 'Stable Branch Tracking' method described in this paper. We deployed it across 4 major versions of FreeBSD.

I would like to thank Julian Elischer for the many conversations that we have had about development method. He provided much of the input into the 'Own Branching' section.

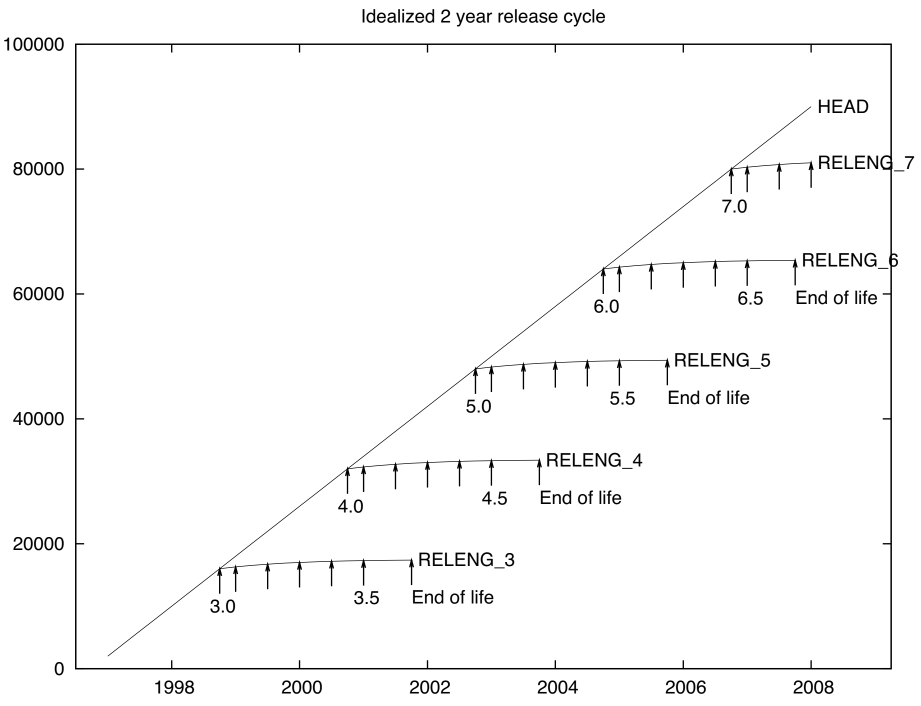

Figure 3: Idealized branching model

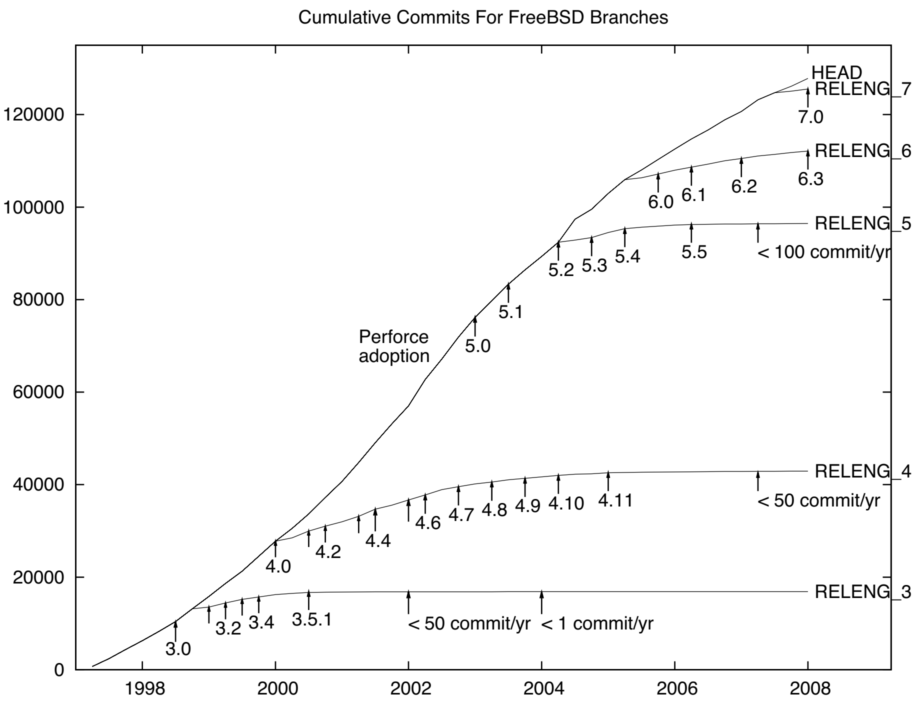

Figure 4: Actual FreeBSD branching history