FreeBSD Release Engineering

Glen Barber, The FreeBSD Foundation Rubicon Communications, LLC (Netgate) <gjb@FreeBSD.org>

2019-07-09 23:59:30 by jhb.

Abstract

This article describes the release engineering process of the FreeBSD Project.

Table of Contents

1. Introduction to the FreeBSD Release Engineering Process

Development of FreeBSD has a very specific workflow. In general, all changes to the FreeBSD base system are committed to the `head/` branch, which reflects the top of the source tree.

After a reasonable testing period, changes can then be merged to the `stable/` branches. The default minimum timeframe before merging to `stable/` branches is three (3) days.

Although a general rule to wait a minimum of three days before merging from `head/`, there are a few special circumstances where an immediate merge may be necessary, such as a critical security fix, or a bug fix that directly inhibits the release build process.

After several months, and the number of changes in the `stable/` branch have grown significantly, it is time to release the next version of FreeBSD. These releases have been historically referred to as "point" releases.

In between releases from the `stable/` branches, approximately every two (2) years, a release will be cut directly from `head/`. These releases have been historically referred to as "dot-zero" releases.

This article will highlight the workflow and responsibilities of the FreeBSD Release Engineering Team for both "dot-zero" and "point"' releases.

The following sections of this article describe:

Section 2, “General Information and Preparation”
: General information and preparation before starting the release cycle.

Section 4, “Website Changes During the Release Cycle”
: Website Changes During the Release Cycle

Section 3, “Release Engineering Terminology”
: Terminology and general information, such as the “code slush” and “code freeze”, used throughout this document.

Section 5, “Release from `head/`”
: The Release Engineering process for a “dot-zero” release.

Section 6, “Release from `stable/`”
: The Release Engineering process for a “point” release.

Section 7, “Building FreeBSD Installation Media”
: Information related to the specific procedures to build installation medium.

Section 8, “Publishing FreeBSD Installation Media to Project Mirrors”
: Procedures to publish installation medium.

Section 9, “Wrapping up the Release Cycle”
: Wrapping up the release cycle.

2. General Information and Preparation

Approximately two months before the start of the release cycle, the FreeBSD Release Engineering Team decides on a schedule for the release. The schedule includes the various milestone points of the release cycle, such as freeze dates, branch dates, and build dates. For example:

Milestone	Anticipated Date
`head/` slush:	May 27, 2016
`head/` freeze:	June 10, 2016
`head/` KBI freeze:	June 24, 2016
`doc/` tree slush [1]:	June 24, 2016
Ports quarterly branch [2]:	July 1, 2016
`stable/`*`12`*`/` branch:	July 8, 2016
`doc/` tree tag [3]:	July 8, 2016
BETA1 build starts:	July 8, 2016
`head/` thaw:	July 9, 2016
BETA2 build starts:	July 15, 2016
BETA3 build starts [*]:	July 22, 2016
`releng/`*`12.0`*`/` branch:	July 29, 2016
RC1 build starts:	July 29, 2016
`stable/`*`12`*`/` thaw:	July 30, 2016
RC2 build starts:	August 5, 2016
Final Ports package builds [4]:	August 6, 2016

Milestone	Anticipated Date
Ports release tag:	August 12, 2016
RC3 build starts [*]:	August 12, 2016
RELEASE build starts:	August 19, 2016
RELEASE announcement:	September 2, 2016

Note

Items marked with "[*]" are "as needed".

1. The `doc/` tree slush is coordinated by the FreeBSD Documentation Engineering Team.
2. The Ports quarterly branch used is determined by when the final `RC` build is planned. A new quarterly branch is created on the first day of the quarter, so this metric should be used when taking the release cycle milestones into account. The quarterly branch is created by the FreeBSD Ports Management Team.
3. The `doc/` tree is tagged by the FreeBSD Documentation Engineering Team.
4. The final Ports package build is done by the FreeBSD Ports Management Team after the final (or what is expected to be final) `RC` build.

Note

If the release is being created from an existing `stable/` branch, the KBI freeze date can be excluded, since the KBI is already considered frozen on established `stable/` branches.

When writing the release cycle schedule, a number of things need to be taken into consideration, in particular milestones where the target date depends on predefined milestones upon which there is a dependency. For example, the Ports Collection release tag originates from the active quarterly branch at the time of the last `RC`. This in part defines which quarterly branch is used, when the release tag can happen, and what revision of the ports tree is used for the final `RELEASE` build.

After general agreement on the schedule, the FreeBSD Release Engineering Team emails the schedule to the FreeBSD Developers.

It is somewhat typical that many developers will inform the FreeBSD Release Engineering Team about various works-in-progress. In some cases, an extension for the in-progress work will be requested, and in other cases, a request for “blanket approval” to a particular subset of the tree will be made.

When such requests are made, it is important to make sure timelines (even if estimated) are discussed. For blanket approvals, the length of time for the blanket approval should be made clear. For example, a FreeBSD developer may request blanket approvals from the start of the code slush until the start of the `RC` builds.

Note

In order to keep track of blanket approvals, the FreeBSD Release Engineering Team uses an internal repository to keep a running log of such requests, which defines the area upon which a blanket approval was granted, the author(s), when the blanket approval expires, and the reason the approval was granted. One example of this is granting blanket approval to `re-`

lease/doc/ to all FreeBSD Release Engineering Team members until the final RC to update the release notes and other release-related documentation.

Note

The FreeBSD Release Engineering Team also uses this repository to track pending approval requests that are received just prior to starting various builds during the release cycle, which the Release Engineer specifies the cutoff period with an email to the FreeBSD developers.

Depending on the underlying set of code in question, and the overall impact the set of code has on FreeBSD as a whole, such requests may be approved or denied by the FreeBSD Release Engineering Team.

The same applies to work-in-progress extensions. For example, in-progress work for a new device driver that is otherwise isolated from the rest of the tree may be granted an extension. A new scheduler, however, may not be feasible, especially if such dramatic changes do not exist in another branch.

The schedule is also added to the Project website, in the doc/ repository, in head/en_US.ISO8859-1/htdocs/releases/*12.0*R/schedule.xml. This file is continuously updated as the release cycle progresses.

Note

In most cases, the schedule.xml can be copied from a prior release and updated accordingly.

In addition to adding schedule.xml to the website, head/share/xml/navibar.ent and head/share/xml/release.ent are also updated to add the link to the schedule to various subpages, as well as enabling the link to the schedule on the Project website index page.

The schedule is also linked from head/en_US.ISO8859-1/htdocs/releng/index.xml .

Approximately one month prior to the scheduled “code slush”, the FreeBSD Release Engineering Team sends a reminder email to the FreeBSD Developers.

Once the first builds of the release cycle are available, update the beta.local.where entity in head/en_US.ISO8859-1/htdocs/releases/ *12.0*R/schedule.xml. replacing IGNORE with INCLUDE.

Note

If two parallel release cycles are happening at once, the beta2.local.where entity may be used instead.

3. Release Engineering Terminology

This section describes some of the terminology used throughout the rest of this document.

3.1. The Code Slush

Although the code slush is not a hard freeze on the tree, the FreeBSD Release Engineering Team requests that bugs in the existing code base take priority over new features.

The code slush does not enforce commit approvals to the branch.

3.2. The Code Freeze

The code freeze marks the point in time where all commits to the branch require explicit approval from the FreeBSD Release Engineering Team.

The FreeBSD Subversion repository contains several hooks to perform sanity checks before any commit is actually committed to the tree. One of these hooks will evaluate if committing to a particular branch requires specific approval.

To enforce commit approvals by the FreeBSD Release Engineering Team, the Release Engineer updates `base/svnadmin/conf/approvers` , and commits the change back to the repository. Once this is done, any change to the branch must include an "Approved by:" line in the commit message.

The "Approved by:" line must match the second column in `base/svnadmin/conf/approvers` , otherwise the commit will be rejected by the repository hooks.

Note

During the code freeze, FreeBSD committers are urged to follow the Change Request Guidelines.

3.3. The KBI/KPI Freeze

KBI/KPI stability implies that the caller of a function across two different releases of software that implement the function results in the same end state. The caller, whether it is a process, thread, or function, expects the function to operate in a certain way, otherwise the KBI/KPI stability on the branch is broken.

4. Website Changes During the Release Cycle

This section describes the changes to the website that should occur as the release cycle progresses.

Note

The files specified throughout this section are relative to the `head/` branch of the `doc` repository in Subversion.

4.1. Website Changes Before the Release Cycle Begins

When the release cycle schedule is available, these files need to be updated to enable various different functionalities on the FreeBSD Project website:

File to Edit	What to Change
`share/xml/release.ent`	Change `beta.upcoming` from `IGNORE` to `INCLUDE`
`share/xml/release.ent`	Change `% beta.upcoming` from `IGNORE` to `INCLUDE`
`share/xml/release.ent`	Change `beta.testing` from `IGNORE` to `INCLUDE`
`share/xml/release.ent`	Change `% beta.testing` from `IGNORE` to `INCLUDE`

4.2. Website Changes During BETA or RC

When transitioning from PRERELEASE to BETA, these files need to be updated to enable the "Help Test" block on the download page. All files are relative to head/ in the doc repository:

File to Edit	What to Change
en_US.ISO8859-1/htdocs/releases/12.0R/schedule.xml	Change % beta.local.where IGNORE to INCLUDE
share/xml/release.ent	Update % betarel.vers to BETA*1*
share/xml/news.xml	Add an entry announcing the BETA

Once the releng/*12.0*/ branch is created, the various release-related documents need to be generated and manually added to the doc/ repository.

Within release/doc, invoke make(1) to generate errata.html, hardware.html, readme.html , and relnotes.html pages, which are then added to doc/head/en_US.ISO8859-1/htdocs/releases/ *X.Y*R/, where *X.Y* represents the major and minor version number of the release.

The fbsd:nokeywords must be set to on on the newly-added files before the pre-commit hooks will allow them to be added to the repository.

4.3. Ports Changes During BETA, RC, and the Final RELEASE

For each build during the release cycle, the MANIFEST files containing the SHA256 of the various distribution sets, such as base.txz , kernel.txz, and so on, are added to the misc/freebsd-release-manifests port. This allows utilities other than bsdinstall(8), such as ports-mgmt/poudriere, to safely use these distribution sets by providing a mechanism through which the checksums can be verified.

5. Release from head/

This section describes the general procedures of the FreeBSD release cycle from the head/ branch.

5.1. FreeBSD "ALPHA" Builds

Starting with the FreeBSD 10.0-RELEASE cycle, the notion of "ALPHA" builds was introduced. Unlike the BETA and RC builds, ALPHA builds are not included in the FreeBSD Release schedule.

The idea behind ALPHA builds is to provide regular FreeBSD-provided builds before the creation of the stable/ branch.

FreeBSD ALPHA snapshots should be built approximately once a week.

For the first ALPHA build, the BRANCH value in sys/conf/newvers.sh needs to be changed from CURRENT to ALPHA1. For subsequent ALPHA builds, increment each ALPHA*N* value by one.

See Section 7, "Building FreeBSD Installation Media" for information on building the ALPHA images.

5.2. Creating the stable/*12*/ Branch

When creating the stable/ branch, several changes are required in both the new stable/ branch and the head/ branch. The files listed are relative to the repository root. To create the new stable/*12*/ branch in Subversion:

```
% svn cp ^/head stable/12/
```

Once the stable/*12*/ branch has been committed, make the following edits:

File to Edit	What to Change
stable/*12*/UPDATING	Update the FreeBSD version, and remove the notice about WITNESS

stable/*12*/contrib/jemalloc/include/jemalloc/jemalloc_FreeBSD.h	`#ifndef MALLOC_PRODUCTION`
	`#define MALLOC_PRODUCTION`
	`#endif`

stable/*12*/lib/clang/llvm.build.mk	Uncomment -DNDEBUG
stable/*12*/sys/*/conf/GENERIC*	Remove debugging support
stable/*12*/sys/*/conf/MINIMAL	Remove debugging support
stable/*12*/release/release.conf.sample	Update SRCBRANCH
stable/*12*/sys/*/conf/GENERIC-NODEBUG	Remove these kernel configurations
stable/*12*/sys/arm/conf/std.arm*	Remove debugging options
stable/*12*/sys/conf/newvers.sh	Update the BRANCH value to reflect BETA1
stable/*12*/share/mk/src.opts.mk	Move REPRODUCIBLE_BUILD from __DEFAULT_NO_OPTIONS to __DEFAULT_YES_OPTIONS
stable/*12*/libexec/rc/rc.conf	Set dumpdev from AUTO to NO (it is configurable via bsdinstall(8) for those that want it enabled by default)
stable/*12*/release/Makefile	Remove the debug.witness.trace entries

Then in the head/ branch, which will now become a new major version:

File to Edit	What to Change
head/UPDATING	Update the FreeBSD version
head/sys/conf/newvers.sh	Update the BRANCH value to reflect CURRENT, and increment REVISION
head/Makefile.inc1	Update TARGET_TRIPLE and MACHINE_TRIPLE
head/sys/sys/param.h	Update __FreeBSD_version
head/gnu/usr.bin/cc/cc_tools/freebsd-native.h	Update FBSD_MAJOR and FBSD_CC_VER
head/contrib/gcc/config.gcc	Append the freebsd<version>.h section
head/lib/clang/llvm.build.mk	Update the value of OS_VERSION
head/lib/clang/freebsd_cc_version.h	Update FREEBSD_CC_VERSION
head/lib/clang/include/lld/Common/Version.inc	Update LLD_REVISION_STRING
head/Makefile.libcompat	Update LILB32CPUFLAGS

6. Release from stable/

This section describes the general procedures of the FreeBSD release cycle from an extablished stable/ branch.

6.1. FreeBSD stable Branch Code Slush

In preparation for the code freeze on a stable branch, several files need to be updated to reflect the release cycle is officially in progress. These files are all relative to the top-most level of the stable branch:

File to Edit	What to Change
sys/conf/newvers.sh	Update the BRANCH value to reflect PRERELEASE
Makefile.inc1	Update TARGET_TRIPLE

File to Edit	What to Change
lib/clang/llvm.build.mk	Update OS_VERSION
gnu/usr.bin/groff/tmac/mdoc.local.in	Add a new .ds entry for the FreeBSD version, and update doc-default-operating-system (FreeBSD 11.x and earlier only)

In the doc repository, also update head/en_US.ISO8859-1/htdocs/releases/ *12.0*R/Makefile.hardware, switching the value of _BRANCH to BETA*X*, RC*X*, or RELEASE, respectively.

6.2. FreeBSD BETA Builds

Following the code slush, the next phase of the release cycle is the code freeze. This is the point at which all commits to the stable branch require explicit approval from the FreeBSD Release Engineering Team. This is enforced by pre-commit hooks in the Subversion repository by editing base/svnadmin/conf/approvers to include a regular expression matching the stable/*12*/ branch for the release:

```
^/stable/12/ re
^/releng/12.0/ re
```

Note

There are two general exceptions to requiring commit approval during the release cycle. The first is any change that needs to be committed by the Release Engineer in order to proceed with the day-to-day workflow of the release cycle, the other is security fixes that may occur during the release cycle.

Once the code freeze is in effect, the next build from the branch is labeled BETA1. This is done by updating the BRANCH value in sys/conf/newvers.sh from PRERELEASE to BETA1.

Once this is done, the first set of BETA builds are started. Subsequent BETA builds do not require updates to any files other than sys/conf/newvers.sh , incrementing the BETA build number.

6.3. Creating the releng/*12.0*/ Branch

When the first RC (Release Candidate) build is ready to begin, the releng/ branch is created. This is a multi-step process that must be done in a specific order, in order to avoid anomalies such as overlaps with __FreeBSD_version values, for example. The paths listed below are relative to the repository root. The order of commits and what to change are:

```
% svn cp ^/stable/12/ releng/12.0/
```

File to Edit	What to Change
releng/*12.0*/sys/conf/newvers.sh	Change BETA*X* to RC1
releng/*12.0*/sys/sys/param.h	Update __FreeBSD_version
releng/*12.0*/etc/pkg/FreeBSD.conf	Replace latest with quarterly as the default package repository location
releng/*12.0*/release/pkg_repos/release-dvd.conf	Replace latest with quarterly as the default package repository location
stable/*12*/sys/conf/newvers.sh	Update BETA*X* with PRERELEASE
stable/*12*/sys/sys/param.h	Update __FreeBSD_version
svnadmin/conf/approvers	Add a new approvers line for the releng branch as was done for the stable branch

```
% svn propdel -R svn:mergeinfo  releng/12.0/
% svn commit  releng/12.0/
% svn commit  stable/12/
```

Now that two new `__FreeBSD_version` values exist, also update `head/en_US.ISO8859-1/books/porters-handbook/versions/chapter.xml` in the Documentation Project repository.

After the first RC build has completed and tested, the `stable/` branch can be "thawed" by removing (or commenting) the `^/stable/12/` entry in `svnadmin/conf/approvers` .

Following the availability of the first RC, FreeBSD Bugmeister Team should be emailed to add the new FreeBSD -RELEASE to the `versions` available in the drop-down menu shown in the bug tracker.

7. Building FreeBSD Installation Media

This section describes the general procedures producing FreeBSD development snapshots and releases.

7.1. Release Build Scripts

This section describes the build scripts used by FreeBSD Release Engineering Team to produce development snapshots and releases.

7.1.1. The `release.sh` Script

Prior to FreeBSD 9.0-RELEASE, `src/release/Makefile` was updated to support bsdinstall(8), and the `src/release/generate-release.sh` script was introduced as a wrapper to automate invoking the release(7) targets.

Prior to FreeBSD 9.2-RELEASE, `src/release/release.sh` was introduced, which heavily based on `src/release/generate-release.sh` included support to specify configuration files to override various options and environment variables. Support for configuration files provided support for cross building each architecture for a release by specifying a separate configuration file for each invocation.

As a brief example of using `src/release/release.sh` to build a single release in `/scratch` :

```
# /bin/sh /usr/src/release/release.sh
```

As a brief example of using `src/release/release.sh` to build a single, cross-built release using a different target directory, create a custom `release.conf` containing:

```
# release.sh configuration for powerpc/powerpc64
CHROOTDIR="/scratch-powerpc64"
TARGET="powerpc"
TARGET_ARCH="powerpc64"
KERNEL="GENERIC64"
```

Then invoke `src/release/release.sh` as:

```
# /bin/sh /usr/src/release/release.sh -c   $HOME/release.conf
```

See release(7) and `src/release/release.conf.sample` for more details and example usage.

7.1.2. The `thermite.sh` Wrapper Script

In order to make cross building the full set of architectures supported on a given branch faster, easier, and reduce human error factors, a wrapper script around `src/release/release.sh` was written to iterate through the various combinations of architectures and invoke `src/release/release.sh` using a configuration file specific to that architecture.

The wrapper script is called `thermite.sh`, which is available in the FreeBSD Subversion repository at `svn://svn.freebsd.org/base/user/gjb/thermite/` , in addition to configuration files used to build `head/` and `stable/12/` development snapshots.

Using `thermite.sh` is covered in Section 7.2, "Building FreeBSD Development Snapshots" and Section 7.3, "Building FreeBSD Releases".

Each architecture and individual kernel have their own configuration file used by `release.sh`. Each branch has its own `defaults-X.conf` configuration which contains entries common throughout each architecture, where overrides or special variables are set and/or overridden in the per-build files.

The per-build configuration file naming scheme is in the form of `${revision}-${TARGET_ARCH}-${KERNCONF}-${type}.conf`, where the uppercase variables are equivalent to what make(1) uses in the build system, and lowercase variables are set within the configuration files, mapping to the major version of the respective branch.

Each branch also has its own `builds-X.conf` configuration, which is used by `thermite.sh`. The `thermite.sh` script iterates through each ${revision}, ${TARGET_ARCH}, ${KERNCONF}, and ${type} value, creating a master list of what to build. However, a given combination from the list will only be built if the respective configuration file exists, which is where the naming scheme above is relevant.

There are two paths of file sourcing:

- `builds-`*`12`*`.conf` -> `main.conf`

  This controls `thermite.sh` behavior

- *`12-amd64-GENERIC-snap`*`.conf` -> `defaults-`*`12`*`.conf` -> `main.conf`

  This controls `release/release.sh` behavior within the build chroot(8)

Note

The `builds-`*`12`*`.conf`, `defaults-`*`12`*`.conf`, and `main.conf` configuration files exist to reduce repetition between the various per-build files.

7.2. Building FreeBSD Development Snapshots

The official release build machines have a specific filesystem layout, which using ZFS, `thermite.sh` takes heavy advantage of with clones and snapshots, ensuring a pristine build environment.

The build scripts reside in `/releng/scripts-snapshot/scripts` or `/releng/scripts-release/scripts` respectively, to avoid collisions between an `RC` build from a releng branch versus a `STABLE` snapshot from the respective stable branch.

A separate dataset exists for the final build images, `/snap/ftp`. This directory contains both snapshots and releases directories. They are only used if the `EVERYTHINGISFINE` variable is defined in `main.conf`.

Note

The `EVERYTHINGISFINE` variable name was chosen to avoid colliding with a variable that might be possibly set in the user environment, accidentally enabling the behavior that depends on it being defined.

As `thermite.sh` iterates through the master list of combinations and locates the per-build configuration file, a ZFS dataset is created under `/releng`, such as `/releng/12-amd64-GENERIC-snap`. The `src/`, `ports/`, and `doc/` trees are checked out to separate ZFS datasets, such as `/releng/12-src-snap`, which are then cloned and mounted into the respective build datasets. This is done to avoid checking out a given tree more than once.

Assuming these filesystem paths, `thermite.sh` would be invoked as:

```
# cd /releng/scripts-snapshot/scripts
# ./setrev.sh -b  stable/12/
# ./zfs-setup.sh -c ./builds-  12.conf
# ./thermite.sh -c ./builds-  12.conf
```

Once the builds have completed, additional helper scripts are available to generate development snapshot emails which are sent to the `freebsd-snapshots@freebsd.org` mailing list:

```
# cd /releng/scripts-snapshot/scripts
# ./get-checksums.sh -c ./builds-  12.conf | ./generate-email.pl > snapshot-  12-mail
```

Note

The generated output should be double-checked for correctness, and the email itself should be PGP signed, in-line.

Note

These helper scripts only apply to development snapshot builds. Announcements during the release cycle (excluding the final release announcement) are created from an email template. A sample of the email template currently used can be found here.

7.3. Building FreeBSD Releases

Similar to building FreeBSD development snapshots, `thermite.sh` would be invoked the same way. The difference between development snapshots and release builds, `BETA` and `RC` included, is that the chroot(8) configuration files must be named with `release` instead of `snap` as the "type", as mentioned above.

In addition, the `BUILDTYPE` and `types` must be changed from `snap` to `release` in `defaults-12.conf` and `builds-12.conf`, respectively.

When building `BETA`, `RC`, and the final `RELEASE`, also statically set `BUILDSVNREV` to the revision on the branch reflecting the name change, `BUILDDATE` to the date the builds are started in `YYYYMMDD` format. If the `doc/` and `ports/` trees have been tagged, also set `PORTBRANCH` and `DOCBRANCH` to the relevant tag path in the Subversion repository, replacing `HEAD` with the last changed revision. Also set `releasesrc` in `builds-12.conf` to the relevant branch, such as `stable/12/` or `releng/12.0/`.

During the release cycle, a copy of `CHECKSUM.SHA512` and `CHECKSUM.SHA256` for each architecture are stored in the FreeBSD Release Engineering Team internal repository in addition to being included in the various announcement emails. Each `MANIFEST` containing the hashes of `base.txz`, `kernel.txz`, etc. are added to misc/freebsd-release-manifests in the Ports Collection, as well.

After building the final `RELEASE`, the `releng/12.0/` branch is tagged as `release/12.0.0/` using the revision from which the `RELEASE` was built. Similar to creating the `stable/12/` and `releng/12.0/` branches, this is done with `svn cp`. From the repository root:

```
% svn cp ^/releng/12.0/@r306420 release/12.0.0/
% svn commit release/12.0.0/
```

8. Publishing FreeBSD Installation Media to Project Mirrors

This section describes the procedure to publish FreeBSD development snapshots and releases to the Project mirrors.

8.1. Staging FreeBSD Installation Media Images

Staging FreeBSD snapshots and releases is a two part process:

- Creating the directory structure to match the hierarchy on `ftp-master`

  If `EVERYTHINGISFINE` is defined in the build configuration files, `main.conf` in the case of the build scripts referenced above, this happens automatically in the chroot(8) after the build is complete, creating the directory structure in `${DESTDIR}/R/ftp-stage` with a path structure matching what is expected on `ftp-master`. This is equivalent to running the following in the chroot(8) directly:

  ```
  # make -C /usr/src/release -f Makefile.mirrors EVERYTHINGISFINE=1 ftp-stage
  ```

  After each architecture is built, `thermite.sh` will rsync the `${DESTDIR}/R/ftp-stage` from the build chroot(8) to `/snap/ftp/snapshots` or `/snap/ftp/releases` on the build host, respectively.

- Copying the files to a staging directory on `ftp-master` before moving the files into `pub/` to begin propagation to the Project mirrors

  Once all builds have finished, `/snap/ftp/snapshots` , or `/snap/ftp/releases` for a release, is polled by `ftp-master` using rsync to `/archive/tmp/snapshots` or `/archive/tmp/releases` , respectively.

> **Note**
>
> On `ftp-master` in the FreeBSD Project infrastructure, this step requires `root` level access, as this step must be executed as the `archive` user.

8.2. Publishing FreeBSD Installation Media

Once the images are staged in `/archive/tmp/` , they are ready to be made public by putting them in `/archive/pub/FreeBSD`. In order to reduce propagation time, pax(1) is used to create hard links from `/archive/tmp` to `/archive/pub/FreeBSD` .

> **Note**
>
> In order for this to be effective, both `/archive/tmp` and `/archive/pub` must reside on the same logical filesystem.

There is a caveat, however, where rsync must be used after pax(1) in order to correct the symbolic links in `pub/FreeBSD/`*`snapshots`*`/ISO-IMAGES` which pax(1) will replace with a hard link, increasing the propagation time.

> **Note**
>
> As with the staging steps, this requires `root` level access, as this step must be executed as the `archive` user.

As the `archive` user:

```
% cd /archive/tmp/ snapshots
% pax -r -w -l . /archive/pub/FreeBSD/  snapshots
```

```
% /usr/local/bin/rsync -avH /archive/tmp/  snapshots /* /archive/pub/FreeBSD/  snapshots /
```

Replace *snapshots* with *releases* as appropriate.

9. Wrapping up the Release Cycle

This section describes general post-release tasks.

9.1. Post-Release Errata Notices

As the release cycle approaches conclusion, it is common to have several EN (Errata Notice) candidates to address issues that were discovered late in the cycle. Following the release, the FreeBSD Release Engineering Team and the FreeBSD Security Team revisit changes that were not approved prior to the final release, and depending on the scope of the change in question, may issue an EN.

Note

The actual process of issuing ENs is handled by the FreeBSD Security Team.

To request an Errata Notice after a release cycle has completed, a developer should fill out the Errata Notice template, in particular the `Background`, `Problem Description`, `Impact`, and if applicable, `Workaround` sections.

The completed Errata Notice template should be emailed together with either a patch against the `releng/` branch or a list of revisions from the `stable/` branch.

For Errata Notice requests immediately following the release, the request should be emailed to both the FreeBSD Release Engineering Team and the FreeBSD Security Team. Once the `releng/` branch has been handed over to the FreeBSD Security Team as described in Section 9.2, “Handoff to the FreeBSD Security Team”, Errata Notice requests should be sent to the FreeBSD Security Team.

9.2. Handoff to the FreeBSD Security Team

Roughly two weeks following the release, the Release Engineer updates `svnadmin/conf/approvers` changing the approver column from `re` to `(so|security-officer)` for the `releng/12.0/` branch.