Committer's Guide

The FreeBSD Documentation Project

Revision: 53619

2019-11-21 17:38:49 by jhb.

Abstract

This document provides information for the FreeBSD committer community. All new committers should read this document before they start, and existing committers are strongly encouraged to review it from time to time.

Almost all FreeBSD developers have commit rights to one or more repositories. However, a few developers do not, and some of the information here applies to them as well. (For instance, some people only have rights to work with the Problem Report database). Please see Section 21, "Issues Specific to Developers Who Are Not Committers" for more information.

This document may also be of interest to members of the FreeBSD community who want to learn more about how the project works.

Table of Contents

1. Administrative Details

Login Methods	ssh(1), protocol 2 only
Main Shell Host	`freefall.FreeBSD.org`
SMTP Host	`smtp.FreeBSD.org:587` (see also Section 6.2.1, "SMTP Access Setup").
`src/` Subversion Root	`svn+ssh://repo.FreeBSD.org/base` (see also Section 5.2.2, "`RELENG_*` Branches and General Layout").
`doc/` Subversion Root	`svn+ssh://repo.FreeBSD.org/doc` (see also Section 5.2.3, "FreeBSD Documentation Project Branches and Layout").
`ports/` Subversion Root	`svn+ssh://repo.FreeBSD.org/ports` (see also Section 5.2.4, "FreeBSD Ports Tree Branches and Layout").
Internal Mailing Lists	developers (technically called all-developers), doc-developers, doc-committers, ports-developers, ports-committers, src-developers, src-committers. (Each project repository has its own -developers and -committers mailing lists. Archives for these lists can be found in the files `/local/mail/`*`repository-name`*`-developers-archive` and `/local/mail/`*`repository-name`*`-committers-archive` on the `FreeBSD.org` cluster.)
Core Team monthly reports	`/home/core/public/monthly-reports` on the `FreeBSD.org` cluster.
Ports Management Team monthly reports	`/home/portmgr/public/monthly-reports` on the `FreeBSD.org` cluster.
Noteworthy `src/` SVN Branches	`stable/`*`n`* (*n*-STABLE), `head` (-CURRENT)

ssh(1) is required to connect to the project hosts. For more information, see Section 16, "SSH Quick-Start Guide".

Useful links:

- FreeBSD Project Internal Pages
- FreeBSD Project Hosts
- FreeBSD Project Administrative Groups

2. OpenPGP Keys for FreeBSD

Cryptographic keys conforming to the OpenPGP (*Pretty Good Privacy*) standard are used by the FreeBSD project to authenticate committers. Messages carrying important information like public SSH keys can be signed with the

OpenPGP key to prove that they are really from the committer. See PGP & GPG: Email for the Practical Paranoid by Michael Lucas and http://en.wikipedia.org/wiki/Pretty_Good_Privacy for more information.

2.1. Creating a Key

Existing keys can be used, but should be checked with `doc/head/share/pgpkeys/checkkey.sh` first. In this case, make sure the key has a FreeBSD user ID.

For those who do not yet have an OpenPGP key, or need a new key to meet FreeBSD security requirements, here we show how to generate one.

1. Install `security/gnupg`. Enter these lines in `~/.gnupg/gpg.conf` to set minimum acceptable defaults:

```
fixed-list-mode
keyid-format 0xlong
personal-digest-preferences SHA512 SHA384 SHA256 SHA224
default-preference-list SHA512 SHA384 SHA256 SHA224 AES256 AES192 AES CAST5 BZIP2 ↺
ZLIB ZIP Uncompressed
use-agent
verify-options show-uid-validity
list-options show-uid-validity
sig-notation issuer-fpr@notations.openpgp.fifthhorseman.net=%g
cert-digest-algo SHA512
```

2. Generate a key:

```
% gpg --full-gen-key
gpg (GnuPG) 2.1.8; Copyright (C) 2015 Free Software Foundation, Inc.
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.

Warning: using insecure memory!
Please select what kind of key you want:
   (1) RSA and RSA (default)
   (2) DSA and Elgamal
   (3) DSA (sign only)
   (4) RSA (sign only)
Your selection? 1
RSA keys may be between 1024 and 4096 bits long.
What keysize do you want? (2048) 2048  ❶
Requested keysize is 2048 bits
Please specify how long the key should be valid.
         0 = key does not expire
      <n>  = key expires in n days
      <n>w = key expires in n weeks
      <n>m = key expires in n months
      <n>y = key expires in n years
Key is valid for? (0) 3y  ❷
Key expires at Wed Nov  4 17:20:20 2015 MST
Is this correct? (y/N) y

GnuPG needs to construct a user ID to identify your key.

Real name: Chucky Daemon  ❸
Email address: notreal@example.com
Comment:
You selected this USER-ID:
    "Chucky Daemon <notreal@example.com>  "

Change (N)ame, (C)omment, (E)mail or (O)kay/(Q)uit? o
You need a Passphrase to protect your secret key.
```

❶ 2048-bit keys with a three-year expiration provide adequate protection at present (2013-12). http://danielpocock.com/rsa-key-sizes-2048-or-4096-bits describes the situation in more detail.

❷ A three year key lifespan is short enough to obsolete keys weakened by advancing computer power, but long enough to reduce key management problems.

❸ Use your real name here, preferably matching that shown on government-issued ID to make it easier for others to verify your identity. Text that may help others identify you can be entered in the `Comment` section.

After the email address is entered, a passphrase is requested. Methods of creating a secure passphrase are contentious. Rather than suggest a single way, here are some links to sites that describe various methods: http://world.std.com/~reinhold/diceware.html, http://www.iusmentis.com/security/passphrasefaq/, http://xkcd.com/936/, http://en.wikipedia.org/wiki/Passphrase.

Protect the private key and passphrase. If either the private key or passphrase may have been compromised or disclosed, immediately notify <accounts@FreeBSD.org> and revoke the key.

Committing the new key is shown in Procedure 1, "Steps for New Committers".

3. Kerberos and LDAP web Password for FreeBSD Cluster

The FreeBSD cluster requires a Kerberos password to access certain services. The Kerberos password also serves as the LDAP web password, since LDAP is proxying to Kerberos in the cluster. Some of the services which require this include:

- Bugzilla
- Jenkins

To create a new Kerberos account in the FreeBSD cluster, or to reset a Kerberos password for an existing account using a random password generator:

```
% ssh kpasswd.freebsd.org
```

Note

This must be done from a machine outside of the FreeBSD.org cluster.

A Kerberos password can also be set manually by logging into `freefall.FreeBSD.org` and running:

```
% kpasswd
```

Note

Unless the Kerberos-authenticated services of the FreeBSD.org cluster have been used previously, Client unknown will be shown. This error means that the `ssh kpasswd.freebsd.org` method shown above must be used first to initialize the Kerberos account.

4. Commit Bit Types

The FreeBSD repository has a number of components which, when combined, support the basic operating system source, documentation, third party application ports infrastructure, and various maintained utilities. When FreeBSD commit bits are allocated, the areas of the tree where the bit may be used are specified. Generally, the areas

associated with a bit reflect who authorized the allocation of the commit bit. Additional areas of authority may be added at a later date: when this occurs, the committer should follow normal commit bit allocation procedures for that area of the tree, seeking approval from the appropriate entity and possibly getting a mentor for that area for some period of time.

Committer Type	*Responsible*	*Tree Components*
src	core@	src/, doc/ subject to appropriate review
doc	doceng@	doc/, ports/, src/ documentation
ports	portmgr@	ports/

Commit bits allocated prior to the development of the notion of areas of authority may be appropriate for use in many parts of the tree. However, common sense dictates that a committer who has not previously worked in an area of the tree seek review prior to committing, seek approval from the appropriate responsible party, and/or work with a mentor. Since the rules regarding code maintenance differ by area of the tree, this is as much for the benefit of the committer working in an area of less familiarity as it is for others working on the tree.

Committers are encouraged to seek review for their work as part of the normal development process, regardless of the area of the tree where the work is occurring.

4.1. Policy for Committer Activity in Other Trees

- All committers may modify `base/head/share/misc/committers-*.dot` , `base/head/usr.bin/calendar/calendars/calendar.freebsd`, and `ports/head/astro/xearth/files` .
- doc committers may commit documentation changes to `src` files, such as man pages, READMEs, fortune databases, calendar files, and comment fixes without approval from a src committer, subject to the normal care and tending of commits.
- Any committer may make changes to any other tree with an "Approved by" from a non-mentored committer with the appropriate bit.
- Committers can acquire an additional bit by the usual process of finding a mentor who will propose them to core, doceng, or portmgr, as appropriate. When approved, they will be added to 'access' and the normal mentoring period will ensue, which will involve a continuing of “Approved by” for some period.
- "Approved by" is only acceptable from non-mentored src committers -- mentored committers can provide a "Reviewed by" but not an "Approved by".

5. Subversion Primer

New committers are assumed to already be familiar with the basic operation of Subversion. If not, start by reading the Subversion Book.

5.1. Introduction

The FreeBSD source repository switched from CVS to Subversion on May 31st, 2008. The first real SVN commit is *r179447*.

The FreeBSD `doc/www` repository switched from CVS to Subversion on May 19th, 2012. The first real SVN commit is *r38821*.

The FreeBSD `ports` repository switched from CVS to Subversion on July 14th, 2012. The first real SVN commit is *r300894*.

Subversion can be installed from the FreeBSD Ports Collection by issuing these commands:

```
# pkg install subversion
```

5.2. Getting Started

There are a few ways to obtain a working copy of the tree from Subversion. This section will explain them.

5.2.1. Direct Checkout

The first is to check out directly from the main repository. For the `src` tree, use:

```
% svn checkout svn+ssh://repo.freebsd.org/base/head /usr/src
```

For the `doc` tree, use:

```
% svn checkout svn+ssh://repo.freebsd.org/doc/head /usr/doc
```

For the `ports` tree, use:

```
% svn checkout svn+ssh://repo.freebsd.org/ports/head /usr/ports
```

Note

Though the remaining examples in this document are written with the workflow of working with the `src` tree in mind, the underlying concepts are the same for working with the `doc` and the `ports` tree. Ports related Subversion operations are listed in Section 20, “Ports Specific FAQ”.

The above command will check out a `CURRENT` source tree as */usr/src/* , which can be any target directory on the local filesystem. Omitting the final argument of that command causes the working copy, in this case, to be named “head”, but that can be renamed safely.

`svn+ssh` means the SVN protocol tunnelled over SSH. The name of the server is `repo.freebsd.org`, `base` is the path to the repository, and `head` is the subdirectory within the repository.

If your FreeBSD login name is different from the login name used on the local machine, either include it in the URL (for example `svn+ssh://jarjar@repo.freebsd.org/base/head`), or add an entry to `~/.ssh/config` in the form:

```
Host repo.freebsd.org
 User jarjar
```

This is the simplest method, but it is hard to tell just yet how much load it will place on the repository.

Note

The `svn diff` does not require access to the server as SVN stores a reference copy of every file in the working copy. This, however, means that Subversion working copies are very large in size.

5.2.2. `RELENG_*` Branches and General Layout

In `svn+ssh://repo.freebsd.org/base` , *base* refers to the source tree. Similarly, *ports* refers to the ports tree, and so on. These are separate repositories with their own change number sequences, access controls and commit mail.

For the base repository, HEAD refers to the -CURRENT tree. For example, `head/bin/ls` is what would go into `/usr/src/bin/ls` in a release. Some key locations are:

- */head/* which corresponds to `HEAD`, also known as `-CURRENT`.
- */stable/n* which corresponds to `RELENG_n`.
- */releng/n.n* which corresponds to `RELENG_n_n`.
- */release/n.n.n* which corresponds to `RELENG_n_n_n_RELEASE`.
- */vendor** is the vendor branch import work area. This directory itself does not contain branches, however its subdirectories do. This contrasts with the *stable*, *releng* and *release* directories.
- */projects* and */user* feature a branch work area. As above, the */user* directory does not contain branches itself.

5.2.3. FreeBSD Documentation Project Branches and Layout

In `svn+ssh://repo.freebsd.org/doc` , *doc* refers to the repository root of the source tree.

In general, most FreeBSD Documentation Project work will be done within the `head/` branch of the documentation source tree.

FreeBSD documentation is written and/or translated to various languages, each in a separate directory in the `head/` branch.

Each translation set contains several subdirectories for the various parts of the FreeBSD Documentation Project. A few noteworthy directories are:

- */articles/* contains the source code for articles written by various FreeBSD contributors.
- */books/* contains the source code for the different books, such as the FreeBSD Handbook.
- */htdocs/* contains the source code for the FreeBSD website.

5.2.4. FreeBSD Ports Tree Branches and Layout

In `svn+ssh://repo.freebsd.org/ports` , *ports* refers to the repository root of the ports tree.

In general, most FreeBSD port work will be done within the `head/` branch of the ports tree which is the actual ports tree used to install software. Some other key locations are:

- */branches/RELENG_n_n_n* which corresponds to `RELENG_n_n_n` is used to merge back security updates in preparation for a release.
- */tags/RELEASE_n_n_n* which corresponds to `RELEASE_n_n_n` represents a release tag of the ports tree.
- */tags/RELEASE_n_EOL* represents the end of life tag of a specific FreeBSD branch.

5.3. Daily Use

This section will explain how to perform common day-to-day operations with Subversion.

5.3.1. Help

SVN has built in help documentation. It can be accessed by typing:

```
% svn help
```

Additional information can be found in the Subversion Book.

5.3.2. Checkout

As seen earlier, to check out the FreeBSD head branch:

```
% svn checkout svn+ssh://repo.freebsd.org/base/head /usr/src
```

At some point, more than just `HEAD` will probably be useful, for instance when merging changes to stable/7. Therefore, it may be useful to have a partial checkout of the complete tree (a full checkout would be very painful).

To do this, first check out the root of the repository:

```
% svn checkout --depth=immediates svn+ssh://repo.freebsd.org/base
```

This will give `base` with all the files it contains (at the time of writing, just `ROADMAP.txt`) and empty subdirectories for `head`, `stable`, `vendor` and so on.

Expanding the working copy is possible. Just change the depth of the various subdirectories:

```
% svn up --set-depth=infinity base/head
% svn up --set-depth=immediates base/release base/releng base/stable
```

The above command will pull down a full copy of `head`, plus empty copies of every `release` tag, every `releng` branch, and every `stable` branch.

If at a later date merging to `7-STABLE` is required, expand the working copy:

```
% svn up --set-depth=infinity base/stable/7
```

Subtrees do not have to be expanded completely. For instance, expanding only `stable/7/sys` and then later expand the rest of `stable/7`:

```
% svn up --set-depth=infinity base/stable/7/sys
% svn up --set-depth=infinity base/stable/7
```

Updating the tree with `svn update` will only update what was previously asked for (in this case, `head` and `stable/7`; it will not pull down the whole tree.

5.3.3. Anonymous Checkout

It is possible to anonymously check out the FreeBSD repository with Subversion. This will give access to a read-only tree that can be updated, but not committed back to the main repository. To do this, use:

```
% svn co https://svn.FreeBSD.org/base/head /usr/src
```

More details on using Subversion this way can be found in Using Subversion.

5.3.4. Updating the Tree

To update a working copy to either the latest revision, or a specific revision:

```
% svn update
% svn update -r12345
```

5.3.5. Status

To view the local changes that have been made to the working copy:

```
% svn status
```

To show local changes and files that are out-of-date do:

```
% svn status --show-updates
```

5.3.6. Editing and Committing

SVN does not need to be told in advance about file editing.

To commit all changes in the current directory and all subdirectories:

```
% svn commit
```

To commit all changes in, for example, *lib/libfetch/* and *usr/bin/fetch/* in a single operation:

```
% svn commit lib/libfetch usr/bin/fetch
```

There is also a commit wrapper for the ports tree to handle the properties and sanity checking the changes:

```
% /usr/ports/Tools/scripts/psvn commit
```

5.3.7. Adding and Removing Files

Note

Before adding files, get a copy of auto-props.txt (there is also a ports tree specific version) and add it to `~/.subversion/config` according to the instructions in the file. If you added something before reading this, use `svn rm --keep-local` for just added files, fix your config file and re-add them again. The initial config file is created when you first run a svn command, even something as simple as `svn help`.

Files are added to a SVN repository with `svn add`. To add a file named *foo*, edit it, then:

```
% svn add foo
```

Note

Most new source files should include a `$FreeBSD$` string near the start of the file. On commit, `svn` will expand the `$FreeBSD$` string, adding the file path, revision number, date and time of commit, and the username of the committer. Files which cannot be modified may be committed without the `$FreeBSD$` string.

Files can be removed with `svn remove`:

```
% svn remove foo
```

Subversion does not require deleting the file before using `svn rm`, and indeed complains if that happens.

It is possible to add directories with `svn add`:

```
% mkdir bar
% svn add bar
```

Although `svn mkdir` makes this easier by combining the creation of the directory and the adding of it:

```
% svn mkdir bar
```

Like files, directories are removed with `svn rm`. There is no separate command specifically for removing directories.

```
% svn rm bar
```

5.3.8. Copying and Moving Files

This command creates a copy of `foo.c` named `bar.c`, with the new file also under version control and with the full history of `foo.c`:

```
% svn copy foo.c bar.c
```

This is usually preferred to copying the file with `cp` and adding it to the repository with `svn add` because this way the new file does not inherit the original one's history.

To move and rename a file:

```
% svn move foo.c bar.c
```

5.3.9. Log and Annotate

`svn log` shows revisions and commit messages, most recent first, for files or directories. When used on a directory, all revisions that affected the directory and files within that directory are shown.

`svn annotate`, or equally `svn praise` or `svn blame`, shows the most recent revision number and who committed that revision for each line of a file.

5.3.10. Diffs

`svn diff` displays changes to the working copy. Diffs generated by SVN are unified and include new files by default in the diff output.

`svn diff` can show the changes between two revisions of the same file:

```
% svn diff -r179453:179454 ROADMAP.txt
```

It can also show all changes for a specific changeset. This command shows what changes were made to the current directory and all subdirectories in changeset 179454:

```
% svn diff -c179454 .
```

5.3.11. Reverting

Local changes (including additions and deletions) can be reverted using `svn revert`. It does not update out-of-date files, but just replaces them with pristine copies of the original version.

5.3.12. Conflicts

If an `svn update` resulted in a merge conflict, Subversion will remember which files have conflicts and refuse to commit any changes to those files until explicitly told that the conflicts have been resolved. The simple, not yet deprecated procedure is:

```
% svn resolved foo
```

However, the preferred procedure is:

```
% svn resolve --accept=working  foo
```

The two examples are equivalent. Possible values for `--accept` are:

- `working`: use the version in your working directory (which one presumes has been edited to resolve the conflicts).
- `base`: use a pristine copy of the version you had before `svn update`, discarding your own changes, the conflicting changes, and possibly other intervening changes as well.
- `mine-full` : use what you had before `svn update`, including your own changes, but discarding the conflicting changes, and possibly other intervening changes as well.
- `theirs-full` : use the version that was retrieved when you did `svn update`, discarding your own changes.

5.4. Advanced Use

5.4.1. Sparse Checkouts

SVN allows *sparse*, or partial checkouts of a directory by adding `--depth` to a `svn checkout`.

Valid arguments to `--depth` are:

- `empty`: the directory itself without any of its contents.
- `files`: the directory and any files it contains.
- `immediates`: the directory and any files and directories it contains, but none of the subdirectories' contents.
- `infinity`: anything.

The `--depth` option applies to many other commands, including `svn commit`, `svn revert`, and `svn diff`.

Since `--depth` is sticky, there is a `--set-depth` option for `svn update` that will change the selected depth. Thus, given the working copy produced by the previous example:

```
% cd ~/freebsd
% svn update --set-depth=immediates .
```

The above command will populate the working copy in *~/freebsd* with `ROADMAP.txt` and empty subdirectories, and nothing will happen when `svn update` is executed on the subdirectories. However, this command will set the depth for *head* (in this case) to infinity, and fully populate it:

```
% svn update --set-depth=infinity   head
```

5.4.2. Direct Operation

Certain operations can be performed directly on the repository without touching the working copy. Specifically, this applies to any operation that does not require editing a file, including:

- `log`, `diff`
- `mkdir`
- `remove`, `copy`, `rename`
- `propset` , `propedit`, `propdel`
- `merge`

Branching is very fast. This command would be used to branch `RELENG_8`:

```
% svn copy svn+ssh://repo.freebsd.org/base/head svn+ssh://repo.freebsd.org/base/stable/8
```

This is equivalent to these commands which take minutes and hours as opposed to seconds, depending on your network connection:

```
% svn checkout --depth=immediates svn+ssh://repo.freebsd.org/base
% cd base
% svn update --set-depth=infinity head
% svn copy head stable/8
% svn commit stable/8
```

5.4.3. Merging with SVN

This section deals with merging code from one branch to another (typically, from head to a stable branch).

Note

In all examples below, `$FSVN` refers to the location of the FreeBSD Subversion repository, `svn+ssh://repo.freebsd.org/base/` .

5.4.3.1. About Merge Tracking

From the user's perspective, merge tracking information (or mergeinfo) is stored in a property called `svn:mergeinfo`, which is a comma-separated list of revisions and ranges of revisions that have been merged. When set on a file, it applies only to that file. When set on a directory, it applies to that directory and its descendants (files and directories) except for those that have their own `svn:mergeinfo`.

It is *not* inherited. For instance, `stable/6/contrib/openpam/` does not implicitly inherit mergeinfo from `stable/6/`, or `stable/6/contrib/`. Doing so would make partial checkouts very hard to manage. Instead, mergeinfo is explicitly propagated down the tree. For merging something into `branch/foo/bar/` , these rules apply:

1. If `branch/foo/bar/` does not already have a mergeinfo record, but a direct ancestor (for instance, `branch/foo/`) does, then that record will be propagated down to `branch/foo/bar/` before information about the current merge is recorded.

2. Information about the current merge will *not* be propagated back up that ancestor.

3. If a direct descendant of `branch/foo/bar/` (for instance, `branch/foo/bar/baz/`) already has a mergeinfo record, information about the current merge will be propagated down to it.

If you consider the case where a revision changes several separate parts of the tree (for example, `branch/foo/bar/` and `branch/foo/quux/`), but you only want to merge some of it (for example, `branch/foo/bar/`), you will see that these rules make sense. If mergeinfo was propagated up, it would seem like that revision had also been merged to `branch/foo/quux/` , when in fact it had not been.

5.4.3.2. Selecting the Source and Target Branch When Merging

Merging to `stable/` branches should originate from `head/`. For example:

```
% svn merge -c r123456 ^/head/ stable/11
% svn commit stable/11
```

Merges to `releng/` branches should always originate from the corresponding `stable/` branch. For example:

```
% svn merge -c r123456 ^/stable/11  releng/11.0
% svn commit releng/11.0
```

Note

Committers are only permitted to commit to the `releng/` branches during a release cycle after receiving approval from the Release Engineering Team, after which only the Security Officer may commit to a `releng/` branch for a Security Advisory or Errata Notice.

All merges are merged to and committed from the root of the branch. All merges look like:

```
% svn merge -c r123456 ^/head/ checkout
% svn commit checkout
```

Note that *`checkout`* must be a complete checkout of the branch to which the merge occurs.

```
% svn merge -c r123456 ^/stable/10 releng/10.0
```

5.4.3.3. Preparing the Merge Target

Because of the mergeinfo propagation issues described earlier, it is very important to never merge changes into a sparse working copy. Always use a full checkout of the branch being merged into. For instance, when merging from HEAD to 7, use a full checkout of stable/7:

```
% cd stable/7
% svn up --set-depth=infinity
```

The target directory must also be up-to-date and must not contain any uncommitted changes or stray files.

5.4.3.4. Identifying Revisions

Identifying revisions to be merged is a must. If the target already has complete mergeinfo, ask SVN for a list:

```
% cd stable/6/contrib/openpam
% svn mergeinfo --show-revs=eligible $FSVN/head/contrib/openpam
```

If the target does not have complete mergeinfo, check the log for the merge source.

5.4.3.5. Merging

Now, let us start merging!

5.4.3.5.1. The Principles

For example, To merge:

- revision $R
- in directory $target in stable branch $B
- from directory $source in head
- $FSVN is `svn+ssh://repo.freebsd.org/base`

Assuming that revisions $P and $Q have already been merged, and that the current directory is an up-to-date working copy of stable/$B, the existing mergeinfo looks like this:

```
% svn propget svn:mergeinfo -R $target
$target - /head/$source:$P,$Q
```

Merging is done like so:

```
% svn merge -c$R $FSVN/head/$source $target
```

Checking the results of this is possible with `svn diff`.

The svn:mergeinfo now looks like:

```
% svn propget svn:mergeinfo -R $target
$target - head/$source:$P,$Q,$R
```

If the results are not exactly as shown, assistance may be required before committing as mistakes may have been made, or there may be something wrong with the existing mergeinfo, or there may be a bug in Subversion.

5.4.3.5.2. Practical Example

As a practical example, consider this scenario. The changes to `netmap.4` in r238987 are to be merged from CURRENT to 9-STABLE. The file resides in `head/share/man/man4` . According to Section 5.4.3, “Merging with SVN”, this is also where to do the merge. Note that in this example all paths are relative to the top of the svn repository. For more information on the directory layout, see Section 5.2.2, “`RELENG_*` Branches and General Layout”.

The first step is to inspect the existing mergeinfo.

```
% svn propget svn:mergeinfo -R stable/9/share/man/man4
```

Take a quick note of how it looks before moving on to the next step; doing the actual merge:

```
% svn merge -c r238987 svn+ssh://repo.freebsd.org/base/head/share/man/man4 stable/9/share/
man/man4
--- Merging r238987 into 'stable/9/share/man/man4':
U    stable/9/share/man/man4/netmap.4
--- Recording mergeinfo for merge of r238987 into
'stable/9/share/man/man4':
 U   stable/9/share/man/man4
```

Check that the revision number of the merged revision has been added. Once this is verified, the only thing left is the actual commit.

```
% svn commit stable/9/share/man/man4
```

5.4.3.6. Precautions Before Committing

As always, build world (or appropriate parts of it).

Check the changes with `svn diff` and `svn stat`. Make sure all the files that should have been added or deleted were in fact added or deleted.

Take a closer look at any property change (marked by a `M` in the second column of `svn stat`). Normally, no svn:mergeinfo properties should be anywhere except the target directory (or directories).

If something looks fishy, ask for help.

5.4.3.7. Committing

Make sure to commit a top level directory to have the mergeinfo included as well. Do not specify individual files on the command line. For more information about committing files in general, see the relevant section of this primer.

5.4.4. Vendor Imports with SVN

Important

Please read this entire section before starting a vendor import.

Note

Patches to vendor code fall into two categories:

- Vendor patches: these are patches that have been issued by the vendor, or that have been extracted from the vendor's version control system, which address issues which cannot wait until the next vendor release.
- FreeBSD patches: these are patches that modify the vendor code to address FreeBSD-specific issues.

The nature of a patch dictates where it should be committed:

- Vendor patches must be committed to the vendor branch, and merged from there to head. If the patch addresses an issue in a new release that is currently being imported, it *must*

not be committed along with the new release: the release must be imported and tagged first, then the patch can be applied and committed. There is no need to re-tag the vendor sources after committing the patch.

- FreeBSD patches are committed directly to head.

5.4.4.1. Preparing the Tree

If importing for the first time after the switch to Subversion, flattening and cleaning up the vendor tree is necessary, as well as bootstrapping the merge history in the main tree.

5.4.4.1.1. Flattening

During the conversion from CVS to Subversion, vendor branches were imported with the same layout as the main tree. This means that the `pf` vendor sources ended up in `vendor/pf/dist/contrib/pf` . The vendor source is best directly in `vendor/pf/dist` .

To flatten the `pf` tree:

```
% cd vendor/pf/dist/contrib/pf
% svn mv $(svn list) ../..
% cd ../..
% svn rm contrib
% svn propdel -R svn:mergeinfo .
% svn commit
```

The `propdel` bit is necessary because starting with 1.5, Subversion will automatically add `svn:mergeinfo` to any directory that is copied or moved. In this case, as nothing is being merged from the deleted tree, they just get in the way.

Tags may be flattened as well (3, 4, 3.5 etc.); the procedure is exactly the same, only changing `dist` to `3.5` or similar, and putting the `svn commit` off until the end of the process.

5.4.4.1.2. Cleaning Up

The `dist` tree can be cleaned up as necessary. Disabling keyword expansion is recommended, as it makes no sense on unmodified vendor code and in some cases it can even be harmful. OpenSSH, for example, includes two files that originated with FreeBSD and still contain the original version tags. To do this:

```
% svn propdel svn:keywords -R .
% svn commit
```

5.4.4.1.3. Bootstrapping Merge History

If importing for the first time after the switch to Subversion, bootstrap `svn:mergeinfo` on the target directory in the main tree to the revision that corresponds to the last related change to the vendor tree, prior to importing new sources:

```
% cd head/contrib/pf
% svn merge --record-only svn+ssh://repo.freebsd.org/base/   vendor/pf/dist@180876   .
% svn commit
```

5.4.4.2. Importing New Sources

With two commits—one for the import itself and one for the tag—this step can optionally be repeated for every upstream release between the last import and the current import.

5.4.4.2.1. Preparing the Vendor Sources

Subversion is able to store a full distribution in the vendor tree. So, import everything, but merge only what is required.

A `svn add` is required to add any files that were added since the last vendor import, and `svn rm` is required to remove any that were removed since. Preparing sorted lists of the contents of the vendor tree and of the sources that are about to be imported is recommended, to facilitate the process.

```
% cd vendor/pf/dist
% svn list -R | grep -v '/$' | sort >../old
% cd ../pf-4.3
% find . -type f | cut -c 3- | sort >../new
```

With these two files, `comm -23 ../old ../new` will list removed files (files only in `old`), while `comm -13 ../old ../new` will list added files only in `new`.

5.4.4.2.2. Importing into the Vendor Tree

Now, the sources must be copied into *`dist`* and the `svn add` and `svn rm` commands are used as needed:

```
% cd vendor/pf/pf-4.3
% tar cf - . | tar xf - -C ../dist
% cd ../dist
% comm -23 ../old ../new | xargs svn rm
% comm -13 ../old ../new | xargs svn add --parents
```

If any directories were removed, they will have to be `svn rm`ed manually. Nothing will break if they are not, but they will remain in the tree.

Check properties on any new files. All text files should have `svn:eol-style` set to `native`. All binary files should have `svn:mime-type` set to `application/octet-stream` unless there is a more appropriate media type. Executable files should have `svn:executable` set to `*`. No other properties should exist on any file in the tree.

Committing is now possible. However, it is good practice to make sure that everything is okay by using the `svn stat` and `svn diff` commands.

5.4.4.2.3. Tagging

Once committed, vendor releases are tagged for future reference. The best and quickest way to do this is directly in the repository:

```
% svn cp svn+ssh://repo.freebsd.org/base/  vendor/pf/dist  svn+ssh://repo.freebsd.org/
base/vendor/pf/4.3
```

Once that is complete, `svn up` the working copy of *`vendor/pf`* to get the new tag, although this is rarely needed.

If creating the tag in the working copy of the tree, `svn:mergeinfo` results must be removed:

```
% cd vendor/pf
% svn cp dist 4.3
% svn propdel svn:mergeinfo -R 4.3
```

5.4.4.3. Merging to Head

```
% cd head/contrib/pf
% svn up
% svn merge --accept=postpone svn+ssh://repo.freebsd.org/base/   vendor/pf/dist  .
```

The `--accept=postpone` tells Subversion not to complain about merge conflicts as they will be handled manually.

Tip

The `cvs2svn` changeover occurred on June 3, 2008. When performing vendor merges for packages which were already present and converted by the `cvs2svn` process, the command used to merge `/vendor/`*`package_name`*`/dist` to `/head/`*`package_location`* (for exam-

ple, `head/contrib/sendmail`) must use `-c REV` to indicate the revision to merge from the `/vendor` tree. For example:

```
% svn checkout svn+ssh://repo.freebsd.org/base/head/contrib/   sendmail
% cd sendmail
% svn merge -c r261190 '^/vendor/sendmail/dist ' .
```

^ is an alias for the repository path.

Note

If using the Zsh shell, the ^ must be escaped with \ or quoted.

It is necessary to resolve any merge conflicts.

Make sure that any files that were added or removed in the vendor tree have been properly added or removed in the main tree. To check diffs against the vendor branch:

```
% svn diff --no-diff-deleted --old=svn+ssh://repo.freebsd.org/base/   vendor/pf/dist  --new=.
```

The `--no-diff-deleted` tells Subversion not to complain about files that are in the vendor tree but not in the main tree. Things that would have previously been removed before the vendor import, like the vendor's makefiles and configure scripts.

Using CVS, once a file was off the vendor branch, it was not able to be put back. With Subversion, there is no concept of on or off the vendor branch. If a file that previously had local modifications, to make it not show up in diffs in the vendor tree, all that has to be done is remove any left-over cruft like FreeBSD version tags, which is much easier.

If any changes are required for the world to build with the new sources, make them now, and keep testing until everything builds and runs perfectly.

5.4.4.4. Committing the Vendor Import

Committing is now possible! Everything must be committed in one go. If done properly, the tree will move from a consistent state with old code, to a consistent state with new code.

5.4.4.5. From Scratch

5.4.4.5.1. Importing into the Vendor Tree

This section is an example of importing and tagging byacc into `head`.

First, prepare the directory in `vendor`:

```
% svn co --depth immediates  $FSVN/vendor
% cd vendor
% svn mkdir byacc
% svn mkdir byacc/dist
```

Now, import the sources into the `dist` directory. Once the files are in place, `svn add` the new ones, then `svn commit` and tag the imported version. To save time and bandwidth, direct remote committing and tagging is possible:

```
% svn cp -m "Tag byacc 20120115"  $FSVN/vendor/byacc/dist  $FSVN/vendor/byacc/20120115
```

5.4.4.5.2. Merging to `head`

Due to this being a new file, copy it for the merge:

```
% svn cp -m "Import byacc to contrib"  $FSVN/vendor/byacc/dist  $FSVN/head/contrib/byacc
```

Working normally on newly imported sources is still possible.

5.4.5. Reverting a Commit

Reverting a commit to a previous version is fairly easy:

```
% svn merge -r179454:179453 ROADMAP.txt
% svn commit
```

Change number syntax, with negative meaning a reverse change, can also be used:

```
% svn merge -c -179454 ROADMAP.txt
% svn commit
```

This can also be done directly in the repository:

```
% svn merge -r179454:179453 svn+ssh://repo.freebsd.org/base/ROADMAP.txt
```

Note

It is important to ensure that the mergeinfo is correct when reverting a file to permit `svn mergeinfo --eligible` to work as expected.

Reverting the deletion of a file is slightly different. Copying the version of the file that predates the deletion is required. For example, to restore a file that was deleted in revision N, restore version N-1:

```
% svn copy svn+ssh://repo.freebsd.org/base/ROADMAP.txt@179454
% svn commit
```

or, equally:

```
% svn copy svn+ssh://repo.freebsd.org/base/ROADMAP.txt@179454 svn+ssh://repo.freebsd.org/
base
```

Do *not* simply recreate the file manually and `svn add` it—this will cause history to be lost.

5.4.6. Fixing Mistakes

While we can do surgery in an emergency, do not plan on having mistakes fixed behind the scenes. Plan on mistakes remaining in the logs forever. Be sure to check the output of `svn status` and `svn diff` before committing.

Mistakes will happen but, they can generally be fixed without disruption.

Take a case of adding a file in the wrong location. The right thing to do is to `svn move` the file to the correct location and commit. This causes just a couple of lines of metadata in the repository journal, and the logs are all linked up correctly.

The wrong thing to do is to delete the file and then `svn add` an independent copy in the correct location. Instead of a couple of lines of text, the repository journal grows an entire new copy of the file. This is a waste.

5.4.7. Using a Subversion Mirror

There is a serious disadvantage to this method: every time something is to be committed, a `svn relocate` to the master repository has to be done, remembering to `svn relocate` back to the mirror after the commit. Also, since `svn relocate` only works between repositories that have the same UUID, some hacking of the local repository's UUID has to occur before it is possible to start using it.

5.4.7.1. Checkout from a Mirror

Check out a working copy from a mirror by substituting the mirror's URL for `svn+ssh://repo.freebsd.org/base` . This can be an official mirror or a mirror maintained by using `svnsync`.

5.4.7.2. Setting up a svnsync Mirror

Avoid setting up a svnsync mirror unless there is a very good reason for it. Most of the time a `git` mirror is a better alternative. Starting a fresh mirror from scratch takes a long time. Expect a minimum of 10 hours for high speed connectivity. If international links are involved, expect this to take four to ten times longer.

One way to limit the time required is to grab a seed file. It is large (~1GB) but will consume less network traffic and take less time to fetch than svnsync will.

Extract the file and update it:

```
% tar xf svnmirror-base-r261170.tar.xz
% svnsync sync file:///home/svnmirror/base
```

Now, set that up to run from cron(8), do checkouts locally, set up a svnserve server for local machines to talk to, etc.

The seed mirror is set to fetch from `svn://svn.freebsd.org/base` . The configuration for the mirror is stored in `revprop 0` on the local mirror. To see the configuration, try:

```
% svn proplist -v --revprop -r 0 file:///home/svnmirror/base
```

Use `svn propset` to change things.

5.4.8. Committing High-ASCII Data

Files that have high-ASCII bits are considered binary files in SVN, so the pre-commit checks fail and indicate that the `mime-type` property should be set to `application/octet-stream` . However, the use of this is discouraged, so please do not set it. The best way is always avoiding high-ASCII data, so that it can be read everywhere with any text editor but if it is not avoidable, instead of changing the mime-type, set the `fbsd:notbinary` property with `propset` :

```
% svn propset fbsd:notbinary yes foo.data
```

5.4.9. Maintaining a Project Branch

A project branch is one that is synced to head (or another branch) is used to develop a project then commit it back to head. In SVN, “dolphin” branching is used for this. A “dolphin” branch is one that diverges for a while and is finally committed back to the original branch. During development code migration in one direction (from head to the branch only). No code is committed back to head until the end. After the branch is committed back at the end, it is dead (although a new branch with the same name can be created after the dead one is deleted).

As per https://people.FreeBSD.org/~peter/svn_notes.txt, work that is intended to be merged back into HEAD should be in `base/projects/` . If the work is beneficial to the FreeBSD community in some way but not intended to be merged directly back into HEAD then the proper location is `base/user/` *`username/`*. This page contains further details.

To create a project branch:

```
% svn copy svn+ssh://repo.freebsd.org/base/head svn+ssh://repo.freebsd.org/base/projects/
spif
```

To merge changes from HEAD back into the project branch:

```
% cd copy_of_spif
% svn merge svn+ssh://repo.freebsd.org/base/head
% svn commit
```

It is important to resolve any merge conflicts before committing.

5.5. Some Tips

In commit logs etc., “rev 179872” is spelled “r179872” as per convention.

Speeding up svn is possible by adding these entries to `~/.ssh/config` :

```
Host *
ControlPath ~/.ssh/sockets/master-%l-%r@%h:%p
ControlMaster auto
ControlPersist yes
```

and then typing

```
mkdir ~/.ssh/sockets
```

Checking out a working copy with a stock Subversion client without FreeBSD-specific patches (`OPTIONS_SET=FREEBSD_TEMPLATE`) will mean that `$FreeBSD$` tags will not be expanded. Once the correct version has been installed, trick Subversion into expanding them like so:

```
% svn propdel -R svn:keywords .
% svn revert -R .
```

This will wipe out uncommitted patches.

It is possible to automatically fill the "Sponsored by" and "MFC after" commit log fields by setting "freebsd-sponsored-by" and "freebsd-mfc-after" fields in the "[miscellany]" section of the `~/.subversion/config` configuration file. For example:

```
freebsd-sponsored-by = The FreeBSD Foundation
freebsd-mfc-after = 2 weeks
```

6. Setup, Conventions, and Traditions

There are a number of things to do as a new developer. The first set of steps is specific to committers only. These steps must be done by a mentor for those who are not committers.

6.1. For New Committers

Those who have been given commit rights to the FreeBSD repositories must follow these steps.

- Get mentor approval before committing each of these changes!
- The `.ent` and `.xml` files mentioned below exist in the FreeBSD Documentation Project SVN repository at `svn+ssh://repo.FreeBSD.org/doc/` .
- New files that do not have the `FreeBSD=%H` `svn:keywords` property will be rejected when attempting to commit them to the repository. Be sure to read Section 5.3.7, “Adding and Removing Files” regarding adding and removing files. Verify that `~/.subversion/config` contains the necessary “auto-props” entries from `auto-props.txt` mentioned there.
- All `src` commits go to FreeBSD-CURRENT first before being merged to FreeBSD-STABLE. The FreeBSD-STABLE branch must maintain ABI and API compatibility with earlier versions of that branch. Do not merge changes that break this compatibility.

Procedure 1. Steps for New Committers

1. Add an Author Entity

   `doc/head/share/xml/authors.ent` — Add an author entity. Later steps depend on this entity, and missing this step will cause the `doc/` build to fail. This is a relatively easy task, but remains a good first test of version control skills.

2. Update the List of Developers and Contributors

   `doc/head/en_US.ISO8859-1/articles/contributors/contrib.committers.xml` — Add an entry to the "Developers" section of the Contributors List. Entries are sorted by last name.

   `doc/head/en_US.ISO8859-1/articles/contributors/contrib.additional.xml` — *Remove* the entry from the "Additional Contributors" section. Entries are sorted by first name.

3. Add a News Item

   `doc/head/share/xml/news.xml` — Add an entry. Look for the other entries that announce new committers and follow the format. Use the date from the commit bit approval email from <core@FreeBSD.org>.

4. Add a PGP Key

   `doc/head/share/pgpkeys/pgpkeys.ent` and `doc/head/share/pgpkeys/pgpkeys-developers.xml` - Add your PGP or GnuPG key. Those who do not yet have a key should see Section 2.1, "Creating a Key".

   Dag-Erling Smørgrav <des@FreeBSD.org> has written a shell script (`doc/head/share/pgpkeys/addkey.sh`) to make this easier. See the README file for more information.

   Use `doc/head/share/pgpkeys/checkkey.sh` to verify that keys meet minimal best-practices standards.

   After adding and checking a key, add both updated files to source control and then commit them. Entries in this file are sorted by last name.

   

   Note

   It is very important to have a current PGP/GnuPG key in the repository. The key may be required for positive identification of a committer. For example, the FreeBSD Administrators <admins@FreeBSD.org> might need it for account recovery. A complete keyring of `FreeBSD.org` users is available for download from https://www.FreeBSD.org/doc/pgpkeyring.txt.

5. Update Mentor and Mentee Information

   `base/head/share/misc/committers-` *`repository`*`.dot` — Add an entry to the current committers section, where *`repository`* is `doc`, `ports`, or `src`, depending on the commit privileges granted.

   Add an entry for each additional mentor/mentee relationship in the bottom section.

6. Generate a Kerberos Password

   See Section 3, "Kerberos and LDAP web Password for FreeBSD Cluster" to generate or set a Kerberos for use with other FreeBSD services like the bug tracking database.

7. Optional: Enable Wiki Account

   FreeBSD Wiki Account — A wiki account allows sharing projects and ideas. Those who do not yet have an account can follow instructions on the AboutWiki Page to obtain one. Contact <wiki-admin@FreeBSD.org> if you need help with your Wiki account.

8. Optional: Update Wiki Information

   Wiki Information - After gaining access to the wiki, some people add entries to the How We Got Here, IRC Nicks, and Dogs of FreeBSD pages.

9. Optional: Update Ports with Personal Information

   `ports/astro/xearth/files/freebsd.committers.markers` and `src/usr.bin/calendar/calendars/calendar.freebsd` - Some people add entries for themselves to these files to show where they are located or the date of their birthday.

10. Optional: Prevent Duplicate Mailings

    Subscribers to svn-src-all, svn-ports-all or svn-doc-all might wish to unsubscribe to avoid receiving duplicate copies of commit messages and followups.

6.2. For Everyone

1. Introduce yourself to the other developers, otherwise no one will have any idea who you are or what you are working on. The introduction need not be a comprehensive biography, just write a paragraph or two about who you are, what you plan to be working on as a developer in FreeBSD, and who will be your mentor. Email this to the FreeBSD developers mailing list and you will be on your way!

2. Log into `freefall.FreeBSD.org` and create a `/var/forward/`*`user`* (where *`user`* is your username) file containing the e-mail address where you want mail addressed to *`yourusername`*@FreeBSD.org to be forwarded. This includes all of the commit messages as well as any other mail addressed to the FreeBSD committer's mailing list and the FreeBSD developers mailing list. Really large mailboxes which have taken up permanent residence on `freefall` may get truncated without warning if space needs to be freed, so forward it or save it elsewhere.

   

   Note

   If your e-mail system uses SPF with strict rules, you should whitelist `mx2.FreeBSD.org` from SPF checks.

   Due to the severe load dealing with SPAM places on the central mail servers that do the mailing list processing, the front-end server does do some basic checks and will drop some messages based on these checks. At the moment proper DNS information for the connecting host is the only check in place but that may change. Some people blame these checks for bouncing valid email. To have these checks turned off for your email, create a file named `~/.spam_lover` on `freefall.FreeBSD.org`.

Note

Those who are developers but not committers will not be subscribed to the committers or developers mailing lists. The subscriptions are derived from the access rights.

6.2.1. SMTP Access Setup

For those willing to send e-mail messages through the FreeBSD.org infrastructure, follow the instructions below:

1. Point your mail client at `smtp.FreeBSD.org:587`.

2. Enable STARTTLS.

3. Ensure your `From:` address is set to *`yourusername`*`@FreeBSD.org`.

4. For authentication, you can use your FreeBSD Kerberos username and password (see Section 3, "Kerberos and LDAP web Password for FreeBSD Cluster"). The *`yourusername`*`/mail` principal is preferred, as it is only valid for authenticating to mail resources.

Note

Do not include `@FreeBSD.org` when entering in your username.

Additional Notes

- Will only accept mail from *yourusername*`@FreeBSD.org`. If you are authenticated as one user, you are not permitted to send mail from another.
- A header will be appended with the SASL username: (`Authenticated sender: `*username*).
- Host has various rate limits in place to cut down on brute force attempts.

6.2.1.1. Using a Local MTA to Forward Emails to the FreeBSD.org SMTP Service

It is also possible to use a local MTA to forward locally sent emails to the FreeBSD.org SMTP servers.

Example 1. Using Postfix

To tell a local Postfix instance that anything from *yourusername*`@FreeBSD.org` should be forwarded to the FreeBSD.org servers, add this to your `main.cf`:

```
sender_dependent_relayhost_maps = hash:/usr/local/etc/postfix/relayhost_maps
smtp_sasl_auth_enable = yes
smtp_sasl_security_options = noanonymous
smtp_sasl_password_maps = hash:/usr/local/etc/postfix/sasl_passwd
smtp_use_tls = yes
```

Create `/usr/local/etc/postfix/relayhost_maps` with the following content:

```
yourusername @FreeBSD.org  [smtp.freebsd.org]:587
```

Create `/usr/local/etc/postfix/sasl_passwd` with the following content:

```
[smtp.freebsd.org]:587           yourusername :yourpassword
```

If the email server is used by other people, you may want to prevent them from sending e-mails from your address. To achieve this, add this to your `main.cf`:

```
smtpd_sender_login_maps = hash:/usr/local/etc/postfix/sender_login_maps
smtpd_sender_restrictions = reject_known_sender_login_mismatch
```

Create `/usr/local/etc/postfix/sender_login_maps` with the following content:

```
yourusername @FreeBSD.org yourlocalusername
```

Where *yourlocalusername* is the SASL username used to connect to the local instance of Postfix.

6.3. Mentors

All new developers have a mentor assigned to them for the first few months. A mentor is responsible for teaching the mentee the rules and conventions of the project and guiding their first steps in the developer community. The mentor is also personally responsible for the mentee's actions during this initial period.

For committers: do not commit anything without first getting mentor approval. Document that approval with an `Approved by:` line in the commit message.

When the mentor decides that a mentee has learned the ropes and is ready to commit on their own, the mentor announces it with a commit to `conf/mentors` . This file is in the `svnadmin` branch of each repository:

`src`	`base/svnadmin/conf/mentors`
`doc`	`doc/svnadmin/conf/mentors`
`ports`	`ports/svnadmin/conf/mentors`

7. Pre-Commit Review

Code review is one way to increase the quality of software. The following guidelines apply to commits to the `head` (-CURRENT) branch of the `src` repository. Other branches and the `ports` and `docs` trees have their own review policies, but these guidelines generally apply to commits requiring review:

- All non-trivial changes should be reviewed before they are committed to the repository.
- Reviews may be conducted by email, in Bugzilla, in Phabricator, or by another mechanism. Where possible, reviews should be public.
- The developer responsible for a code change is also responsible for making all necessary review-related changes.
- Code review can be an iterative process, which continues until the patch is ready to be committed. Specifically, once a patch is sent out for review, it should receive an explicit “looks good” before it is committed. So long as it is explicit, this can take whatever form makes sense for the review method.
- Timeouts are not a substitute for review.

Sometimes code reviews will take longer than you would hope for, especially for larger features. Accepted ways to speed up review times for your patches are:

- Review other people's patches. If you help out, everybody will be more willing to do the same for you; goodwill is our currency.
- Ping the patch. If it is urgent, provide reasons why it is important to you to get this patch landed and ping it every couple of days. If it is not urgent, the common courtesy ping rate is one week. Remember that you are asking for valuable time from other professional developers.
- Ask for help on mailing lists, IRC, etc. Others may be able to either help you directly, or suggest a reviewer.
- Split your patch into multiple smaller patches that build on each other. The smaller your patch, the higher the probability that somebody will take a quick look at it.

  When making large changes, it is helpful to keep this in mind from the beginning of the effort as breaking large changes into smaller ones is often difficult after the fact.

Developers should participate in code reviews as both reviewers and reviewees. If someone is kind enough to review your code, you should return the favor for someone else. Note that while anyone is welcome to review and

give feedback on a patch, only an appropriate subject-matter expert can approve a change. This will usually be a committer who works with the code in question on a regular basis.

In some cases, no subject-matter expert may be available. In those cases, a review by an experienced developer is sufficient when coupled with appropriate testing.

8. Commit Log Messages

This section contains some suggestions and traditions for how commit logs are formatted.

As well as including an informative message with each commit, some additional information may be needed.

This information consists of one or more lines containing the key word or phrase, a colon, tabs for formatting, and then the additional information.

The key words or phrases are:

`PR:`	The problem report (if any) which is affected (typically, by being closed) by this commit. Multiple PRs may be specified on one line, separated by commas or spaces.
`Submitted by:`	The name and e-mail address of the person that submitted the fix; for developers, just the username on the FreeBSD cluster. If the submitter is the maintainer of the port being committed, include "(maintainer)" after the email address. Avoid obfuscating the email address of the submitter as this adds additional work when searching logs.
`Reviewed by:`	The name and e-mail address of the person or people that reviewed the change; for developers, just the username on the FreeBSD cluster. If a patch was submitted to a mailing list for review, and the review was favorable, then just include the list name.

`Approved by:`	The name and e-mail address of the person or people that approved the change; for developers, just the username on the FreeBSD cluster. It is customary to get prior approval for a commit if it is to an area of the tree to which you do not usually commit. In addition, during the run up to a new release all commits *must* be approved by the release engineering team.
	
	While under mentorship, get mentor approval before the commit. Enter the mentor's username in this field, and note that they are a mentor:
	
	`Approved by: username-of-mentor (mentor)`
	
	If a team approved these commits then include the team name followed by the username of the approver in parentheses. For example:
	
	`Approved by: re (username)`

`Obtained from:`	The name of the project (if any) from which the code was obtained. Do not use this line for the name of an individual person.

`Sponsored by:`	Sponsoring organizations for this change, if any. Separate multiple organizations with commas. If only a portion of the work was sponsored, or different amounts of sponsorship were provided to different authors, please give appropriate credit in parentheses after each sponsor name. For example, `Example.com (alice, code refactoring), Wormulon (bob), Momcorp (cindy)` shows that Alice was sponsored by Example.com to do code refactoring, while Wormulon sponsored Bob's work and Momcorp sponsored Cindy's work. Other authors were either not sponsored or chose not to list sponsorship.
`MFC after:`	To receive an e-mail reminder to MFC at a later date, specify the number of days, weeks, or months after which an MFC is planned.
`MFC to:`	If the commit should be merged to a subset of stable branches, specify the branch names.
`MFC with:`	If the commit should be merged together with a previous one in a single MFC commit (for example, where this commit corrects a bug in the previous change), specify the corresponding revision number.
`Relnotes:`	If the change is a candidate for inclusion in the release notes for the next release from the branch, set to `yes`.
`Security:`	If the change is related to a security vulnerability or security exposure, include one or more references or a description of the issue. If possible, include a VuXML URL or a CVE ID.
`Event:`	The description for the event where this commit was made. If this is a recurring event, add the year or even the month to it. For example, this could be `FooBSDcon 2019`. The idea behind this line is to put recognition to conferences, gatherings, and other types of meetups and to show that these are useful to have. Please do not use the `Sponsored by:` line for this as that is meant for organizations sponsoring certain features or developers working on them.
`Differential Revision:`	The full URL of the Phabricator review. This line *must be the last line*. For example: `https://reviews.freebsd.org/D1708`.

Example 2. Commit Log for a Commit Based on a PR

The commit is based on a patch from a PR submitted by John Smith. The commit message "PR" and "Submitted by" fields are filled..

```
...

     PR:                     12345
     Submitted by:    John Smith <John.Smith@example.com>
```

Example 3. Commit Log for a Commit Needing Review

The virtual memory system is being changed. After posting patches to the appropriate mailing list (in this case, `freebsd-arch`) and the changes have been approved.

```
...
     Reviewed by:       -arch
```

Example 4. Commit Log for a Commit Needing Approval

Commit a port, after working with the listed MAINTAINER, who said to go ahead and commit.

```
...
     Approved by:     abc (maintainer)
```

Where *`abc`* is the account name of the person who approved.

Example 5. Commit Log for a Commit Bringing in Code from OpenBSD

Committing some code based on work done in the OpenBSD project.

```
...
     Obtained from:      OpenBSD
```

Example 6. Commit Log for a Change to FreeBSD-CURRENT with a Planned Commit to FreeBSD-STABLE to Follow at a Later Date.

Committing some code which will be merged from FreeBSD-CURRENT into the FreeBSD-STABLE branch after two weeks.

```
...
MFC after:      2 weeks
```

Where *`2`* is the number of days, weeks, or months after which an MFC is planned. The *`weeks`* option may be `day`, `days`, `week`, `weeks`, `month`, `months`.

It is often necessary to combine these.

Consider the situation where a user has submitted a PR containing code from the NetBSD project. Looking at the PR, the developer sees it is not an area of the tree they normally work in, so they have the change reviewed by the `arch` mailing list. Since the change is complex, the developer opts to MFC after one month to allow adequate testing.

The extra information to include in the commit would look something like

Example 7. Example Combined Commit Log

```
PR:                 54321
Submitted by:       John Smith <John.Smith@example.com>
Reviewed by:        -arch
Obtained from:      NetBSD
MFC after:          1 month
Relnotes:           yes
```

9. Preferred License for New Files

The FreeBSD Project's full license policy can be found at https://www.FreeBSD.org/internal/software-license.html. The rest of this section is intended to help you get started. As a rule, when in doubt, ask. It is much easier to give advice than to fix the source tree.

The FreeBSD Project suggests and uses this text as the preferred license scheme:

```
/*-
 * SPDX-License-Identifier: BSD-2-Clause-FreeBSD
 *
 * Copyright (c) [year] [your name]
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in the
 *    documentation and/or other materials provided with the distribution.
 *
 * THIS SOFTWARE IS PROVIDED BY THE AUTHOR AND CONTRIBUTORS ``AS IS'' AND
 * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED.  IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE
 * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
 * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 * OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
 * OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
 * SUCH DAMAGE.
 *
 * [id for your version control system, if any]
 */
```

The FreeBSD project strongly discourages the so-called "advertising clause" in new code. Due to the large number of contributors to the FreeBSD project, complying with this clause for many commercial vendors has become difficult. If you have code in the tree with the advertising clause, please consider removing it. In fact, please consider using the above license for your code.

The FreeBSD project discourages completely new licenses and variations on the standard licenses. New licenses require the approval of the Core Team <core@FreeBSD.org> to reside in the main repository. The more different licenses that are used in the tree, the more problems that this causes to those wishing to utilize this code, typically from unintended consequences from a poorly worded license.

Project policy dictates that code under some non-BSD licenses must be placed only in specific sections of the repository, and in some cases, compilation must be conditional or even disabled by default. For example, the GENERIC kernel must be compiled under only licenses identical to or substantially similar to the BSD license. GPL, APSL, CDDL, etc, licensed software must not be compiled into GENERIC.

Developers are reminded that in open source, getting "open" right is just as important as getting "source" right, as improper handling of intellectual property has serious consequences. Any questions or concerns should immediately be brought to the attention of the core team.

10. Keeping Track of Licenses Granted to the FreeBSD Project

Various software or data exist in the repositories where the FreeBSD project has been granted a special licence to be able to use them. A case in point are the Terminus fonts for use with vt(4). Here the author Dimitar Zhekov has allowed us to use the "Terminus BSD Console" font under a 2-clause BSD license rather than the regular Open Font License he normally uses.

It is clearly sensible to keep a record of any such license grants. To that end, the Core Team <core@FreeBSD.org> has decided to keep an archive of them. Whenever the FreeBSD project is granted a special license we require the Core Team <core@FreeBSD.org> to be notified. Any developers involved in arranging such a license grant, please send details to the Core Team <core@FreeBSD.org> including:

- Contact details for people or organizations granting the special license.
- What files, directories etc. in the repositories are covered by the license grant including the revision numbers where any specially licensed material was committed.
- The date the license comes into effect from. Unless otherwise agreed, this will be the date the license was issued by the authors of the software in question.
- The license text.
- A note of any restrictions, limitations or exceptions that apply specifically to FreeBSD's usage of the licensed material.
- Any other relevant information.

Once the Core Team <core@FreeBSD.org> is satisfied that all the necessary details have been gathered and are correct, the secretary will send a PGP-signed acknowledgement of receipt including the license details. This receipt will be persistently archived and serve as our permanent record of the license grant.

The license archive should contain only details of license grants; this is not the place for any discussions around licensing or other subjects. Access to data within the license archive will be available on request to the Core Team <core@FreeBSD.org>.

11. Developer Relations

When working directly on your own code or on code which is already well established as your responsibility, then there is probably little need to check with other committers before jumping in with a commit. Working on a bug in an area of the system which is clearly orphaned (and there are a few such areas, to our shame), the same applies. When modifying parts of the system which are maintained, formally, or informally, consider asking for review just as a developer would have before becoming a committer. For ports, contact the listed `MAINTAINER` in the `Makefile`.

To determine if an area of the tree is maintained, check the MAINTAINERS file at the root of the tree. If nobody is listed, scan the revision history to see who has committed changes in the past. An example script that lists each person who has committed to a given file along with the number of commits each person has made can be found at on `freefall` at `~eadler/bin/whodid .` If queries go unanswered or the committer otherwise indicates a lack of interest in the area affected, go ahead and commit it.

Important

Avoid sending private emails to maintainers. Other people might be interested in the conversation, not just the final output.

If there is any doubt about a commit for any reason at all, have it reviewed before committing. Better to have it flamed then and there rather than when it is part of the repository. If a commit does results in controversy erupting, it may be advisable to consider backing the change out again until the matter is settled. Remember, with a version control system we can always change it back.

Do not impugn the intentions of others. If they see a different solution to a problem, or even a different problem, it is probably not because they are stupid, because they have questionable parentage, or because they are trying to destroy hard work, personal image, or FreeBSD, but basically because they have a different outlook on the world. Different is good.

Disagree honestly. Argue your position from its merits, be honest about any shortcomings it may have, and be open to seeing their solution, or even their vision of the problem, with an open mind.

Accept correction. We are all fallible. When you have made a mistake, apologize and get on with life. Do not beat up yourself, and certainly do not beat up others for your mistake. Do not waste time on embarrassment or recrimination, just fix the problem and move on.

Ask for help. Seek out (and give) peer reviews. One of the ways open source software is supposed to excel is in the number of eyeballs applied to it; this does not apply if nobody will review code.

12. If in Doubt...

When unsure about something, whether it be a technical issue or a project convention be sure to ask. If you stay silent you will never make progress.

If it relates to a technical issue ask on the public mailing lists. Avoid the temptation to email the individual person that knows the answer. This way everyone will be able to learn from the question and the answer.

For project specific or administrative questions ask, in order:

- Your mentor or former mentor.
- An experienced committer on IRC, email, etc.
- Any team with a "hat", as they can give you a definitive answer.
- If still not sure, ask on FreeBSD developers mailing list.

Once your question is answered, if no one pointed you to documentation that spelled out the answer to your question, document it, as others will have the same question.

13. Bugzilla

The FreeBSD Project utilizes Bugzilla for tracking bugs and change requests. Be sure that if you commit a fix or suggestion found in the PR database to close it. It is also considered nice if you take time to close any PRs associated with your commits, if appropriate.

Committers with non-`FreeBSD.org` Bugzilla accounts can have the old account merged with the `FreeBSD.org` account by following these steps:

1. Log in using your old account.

2. Open new bug. Choose `Services` as the Product, and `Bug Tracker` as the Component. In bug description list acounts you wish to be merged.

3. Log in using `FreeBSD.org` account and post comment to newly opened bug to confirm ownership. See Section 3, "Kerberos and LDAP web Password for FreeBSD Cluster" for more details on how to generate or set a password for your `FreeBSD.org` account.

4. If there are more than two accounts to merge, post comments from each of them.

You can find out more about Bugzilla at:

- FreeBSD Problem Report Handling Guidelines
- https://www.FreeBSD.org/support.html

14. Phabricator

The FreeBSD Project utilizes Phabricator for code review requests. See the CodeReview wiki page for details.

Committers with non-`FreeBSD.org` Phabricator accounts can have the old account renamed to the `FreeBSD.org` account by following these steps:

1. Change your Phabricator account email to your `FreeBSD.org` email.

2. Open new bug on our bug tracker using your `FreeBSD.org` account, see Section 13, "Bugzilla" for more information. Choose `Services` as the Product, and `Code Review` as the Component. In bug description request that your Phabricator account be renamed, and provide a link to your Phabricator user. For example, `https://reviews.freebsd.org/p/`*`bob_example.com`*`/`

Important

Phabricator accounts cannot be merged, please do not open a new account.

15. Who's Who

Besides the repository meisters, there are other FreeBSD project members and teams whom you will probably get to know in your role as a committer. Briefly, and by no means all-inclusively, these are:

Documentation Engineering Team <doceng@FreeBSD.org>
: doceng is the group responsible for the documentation build infrastructure, approving new documentation committers, and ensuring that the FreeBSD website and documentation on the FTP site is up to date with respect to the subversion tree. It is not a conflict resolution body. The vast majority of documentation related discussion takes place on the FreeBSD documentation project mailing list. More details regarding the doceng team can be found in its charter. Committers interested in contributing to the documentation should familiarize themselves with the Documentation Project Primer.

Bruce Evans <bde@FreeBSD.org>
: Bruce is the Style Police-Meister. When you do a commit that could have been done better, Bruce will be there to tell you. Be thankful that someone is. Bruce is also very knowledgeable on the various standards applicable to FreeBSD.

Glen Barber <gjb@FreeBSD.org>, Konstantin Belousov <kib@FreeBSD.org>, Bryan Drewery <bdrewery@FreeBSD.org>, Marc Fonvieille <blackend@FreeBSD.org>, Xin Li <delphij@FreeBSD.org>, Hiroki Sato <hrs@FreeBSD.org>, Gleb Smirnoff <glebius@FreeBSD.org>, Marius Strobl <marius@FreeBSD.org>

These are the members of the Release Engineering Team <re@FreeBSD.org>. This team is responsible for setting release deadlines and controlling the release process. During code freezes, the release engineers have final authority on all changes to the system for whichever branch is pending release status. If there is something you want merged from FreeBSD-CURRENT to FreeBSD-STABLE (whatever values those may have at any given time), these are the people to talk to about it.

Gordon Tetlow <gordon@FreeBSD.org>

Gordon Tetlow is the FreeBSD Security Officer and oversees the Security Officer Team <security-officer@FreeBSD.org>.

Garrett Wollman <wollman@FreeBSD.org>

If you need advice on obscure network internals or are not sure of some potential change to the networking subsystem you have in mind, Garrett is someone to talk to. Garrett is also very knowledgeable on the various standards applicable to FreeBSD.

FreeBSD committer's mailing list

svn-src-all, svn-ports-all and svn-doc-all are the mailing lists that the version control system uses to send commit messages to. *Never* send email directly to these lists. Only send replies to this list when they are short and are directly related to a commit.

FreeBSD developers mailing list

All committers are subscribed to -developers. This list was created to be a forum for the committers "community" issues. Examples are Core voting, announcements, etc.

The FreeBSD developers mailing list is for the exclusive use of FreeBSD committers. To develop FreeBSD, committers must have the ability to openly discuss matters that will be resolved before they are publicly announced. Frank discussions of work in progress are not suitable for open publication and may harm FreeBSD.

All FreeBSD committers are expected not to not publish or forward messages from the FreeBSD developers mailing list outside the list membership without permission of all of the authors. Violators will be removed from the FreeBSD developers mailing list, resulting in a suspension of commit privileges. Repeated or flagrant violations may result in permanent revocation of commit privileges.

This list is *not* intended as a place for code reviews or for any technical discussion. In fact using it as such hurts the FreeBSD Project as it gives a sense of a closed list where general decisions affecting all of the FreeBSD using community are made without being "open". Last, but not least *never, never ever, email the FreeBSD developers mailing list and CC:/BCC: another FreeBSD list*. Never, ever email another FreeBSD email list and CC:/BCC: the FreeBSD developers mailing list. Doing so can greatly diminish the benefits of this list.

16. SSH Quick-Start Guide

1. If you do not wish to type your password in every time you use ssh(1), and you use keys to authenticate, ssh-agent(1) is there for your convenience. If you want to use ssh-agent(1), make sure that you run it before running other applications. X users, for example, usually do this from their `.xsession` or `.xinitrc`. See ssh-agent(1) for details.

2. Generate a key pair using ssh-keygen(1). The key pair will wind up in your `$HOME/.ssh/` directory.

Important

Only ECDSA, Ed25519 or RSA keys are supported.

3. Send your public key (`$HOME/.ssh/id_ecdsa.pub` , `$HOME/.ssh/id_ed25519.pub` , or `$HOME/.ssh/id_rsa.pub`) to the person setting you up as a committer so it can be put into *`yourlogin`* in `/etc/ssh-keys/` on `freefall`.

Now ssh-add(1) can be used for authentication once per session. It prompts for the private key's pass phrase, and then stores it in the authentication agent (ssh-agent(1)). Use `ssh-add -d` to remove keys stored in the agent.

Test with a simple remote command: `ssh freefall.FreeBSD.org ls /usr`.

For more information, see security/openssh, ssh(1), ssh-add(1), ssh-agent(1), ssh-keygen(1), and scp(1).

For information on adding, changing, or removing ssh(1) keys, see `this article`.

17. Coverity® Availability for FreeBSD Committers

All FreeBSD developers can obtain access to Coverity analysis results of all FreeBSD Project software. All who are interested in obtaining access to the analysis results of the automated Coverity runs, can sign up at `Coverity Scan`.

The FreeBSD wiki includes a mini-guide for developers who are interested in working with the Coverity® analysis reports: `https://wiki.freebsd.org/CoverityPrevent` . Please note that this mini-guide is only readable by FreeBSD developers, so if you cannot access this page, you will have to ask someone to add you to the appropriate Wiki access list.

Finally, all FreeBSD developers who are going to use Coverity® are always encouraged to ask for more details and usage information, by posting any questions to the mailing list of the FreeBSD developers.

18. The FreeBSD Committers' Big List of Rules

Everyone involved with the FreeBSD project is expected to abide by the *Code of Conduct* available from https://www.FreeBSD.org/internal/code-of-conduct.html. As committers, you form the public face of the project, and how you behave has a vital impact on the public perception of it. This guide expands on the parts of the *Code of Conduct* specific to committers.

1. Respect other committers.

2. Respect other contributors.

3. Discuss any significant change *before* committing.

4. Respect existing maintainers (if listed in the `MAINTAINER` field in `Makefile` or in `MAINTAINER` in the top-level directory).

5. Any disputed change must be backed out pending resolution of the dispute if requested by a maintainer. Security related changes may override a maintainer's wishes at the Security Officer's discretion.

6. Changes go to FreeBSD-CURRENT before FreeBSD-STABLE unless specifically permitted by the release engineer or unless they are not applicable to FreeBSD-CURRENT. Any non-trivial or non-urgent change which is applic-

able should also be allowed to sit in FreeBSD-CURRENT for at least 3 days before merging so that it can be given sufficient testing. The release engineer has the same authority over the FreeBSD-STABLE branch as outlined for the maintainer in rule #5.

7. Do not fight in public with other committers; it looks bad.

8. Respect all code freezes and read the `committers` and `developers` mailing lists in a timely manner so you know when a code freeze is in effect.

9. When in doubt on any procedure, ask first!

10. Test your changes before committing them.

11. Do not commit to contributed software without *explicit* approval from the respective maintainers.

As noted, breaking some of these rules can be grounds for suspension or, upon repeated offense, permanent removal of commit privileges. Individual members of core have the power to temporarily suspend commit privileges until core as a whole has the chance to review the issue. In case of an "emergency" (a committer doing damage to the repository), a temporary suspension may also be done by the repository meisters. Only a 2/3 majority of core has the authority to suspend commit privileges for longer than a week or to remove them permanently. This rule does not exist to set core up as a bunch of cruel dictators who can dispose of committers as casually as empty soda cans, but to give the project a kind of safety fuse. If someone is out of control, it is important to be able to deal with this immediately rather than be paralyzed by debate. In all cases, a committer whose privileges are suspended or revoked is entitled to a "hearing" by core, the total duration of the suspension being determined at that time. A committer whose privileges are suspended may also request a review of the decision after 30 days and every 30 days thereafter (unless the total suspension period is less than 30 days). A committer whose privileges have been revoked entirely may request a review after a period of 6 months has elapsed. This review policy is *strictly informal* and, in all cases, core reserves the right to either act on or disregard requests for review if they feel their original decision to be the right one.

In all other aspects of project operation, core is a subset of committers and is bound by the *same rules*. Just because someone is in core this does not mean that they have special dispensation to step outside any of the lines painted here; core's "special powers" only kick in when it acts as a group, not on an individual basis. As individuals, the core team members are all committers first and core second.

18.1. Details

1. Respect other committers.

   This means that you need to treat other committers as the peer-group developers that they are. Despite our occasional attempts to prove the contrary, one does not get to be a committer by being stupid and nothing rankles more than being treated that way by one of your peers. Whether we always feel respect for one another or not (and everyone has off days), we still have to *treat* other committers with respect at all times, on public forums and in private email.

   Being able to work together long term is this project's greatest asset, one far more important than any set of changes to the code, and turning arguments about code into issues that affect our long-term ability to work harmoniously together is just not worth the trade-off by any conceivable stretch of the imagination.

   To comply with this rule, do not send email when you are angry or otherwise behave in a manner which is likely to strike others as needlessly confrontational. First calm down, then think about how to communicate in the most effective fashion for convincing the other persons that your side of the argument is correct, do not just blow off some steam so you can feel better in the short term at the cost of a long-term flame war. Not only is this very bad "energy economics", but repeated displays of public aggression which impair our ability to work well together will be dealt with severely by the project leadership and may result in suspension or termination of your commit privileges. The project leadership will take into account both public and private communications brought before it. It will not seek the disclosure of private communications, but it will take it into account if it is volunteered by the committers involved in the complaint.

All of this is never an option which the project's leadership enjoys in the slightest, but unity comes first. No amount of code or good advice is worth trading that away.

2. Respect other contributors.

   You were not always a committer. At one time you were a contributor. Remember that at all times. Remember what it was like trying to get help and attention. Do not forget that your work as a contributor was very important to you. Remember what it was like. Do not discourage, belittle, or demean contributors. Treat them with respect. They are our committers in waiting. They are every bit as important to the project as committers. Their contributions are as valid and as important as your own. After all, you made many contributions before you became a committer. Always remember that.

   Consider the points raised under 1 and apply them also to contributors.

3. Discuss any significant change *before* committing.

   The repository is not where changes are initially submitted for correctness or argued over, that happens first in the mailing lists or by use of the Phabricator service. The commit will only happen once something resembling consensus has been reached. This does not mean that permission is required before correcting every obvious syntax error or manual page misspelling, just that it is good to develop a feel for when a proposed change is not quite such a no-brainer and requires some feedback first. People really do not mind sweeping changes if the result is something clearly better than what they had before, they just do not like being *surprised* by those changes. The very best way of making sure that things are on the right track is to have code reviewed by one or more other committers.

   When in doubt, ask for review!

4. Respect existing maintainers if listed.

   Many parts of FreeBSD are not "owned" in the sense that any specific individual will jump up and yell if you commit a change to "their" area, but it still pays to check first. One convention we use is to put a maintainer line in the `Makefile` for any package or subtree which is being actively maintained by one or more people; see https://www.FreeBSD.org/doc/en_US.ISO8859-1/books/developers-handbook/policies.html for documentation on this. Where sections of code have several maintainers, commits to affected areas by one maintainer need to be reviewed by at least one other maintainer. In cases where the "maintainer-ship" of something is not clear, look at the repository logs for the files in question and see if someone has been working recently or predominantly in that area.

5. Any disputed change must be backed out pending resolution of the dispute if requested by a maintainer. Security related changes may override a maintainer's wishes at the Security Officer's discretion.

   This may be hard to swallow in times of conflict (when each side is convinced that they are in the right, of course) but a version control system makes it unnecessary to have an ongoing dispute raging when it is far easier to simply reverse the disputed change, get everyone calmed down again and then try to figure out what is the best way to proceed. If the change turns out to be the best thing after all, it can be easily brought back. If it turns out not to be, then the users did not have to live with the bogus change in the tree while everyone was busily debating its merits. People *very* rarely call for back-outs in the repository since discussion generally exposes bad or controversial changes before the commit even happens, but on such rare occasions the back-out should be done without argument so that we can get immediately on to the topic of figuring out whether it was bogus or not.

6. Changes go to FreeBSD-CURRENT before FreeBSD-STABLE unless specifically permitted by the release engineer or unless they are not applicable to FreeBSD-CURRENT. Any non-trivial or non-urgent change which is applicable should also be allowed to sit in FreeBSD-CURRENT for at least 3 days before merging so that it can be given sufficient testing. The release engineer has the same authority over the FreeBSD-STABLE branch as outlined in rule #5.

This is another “do not argue about it” issue since it is the release engineer who is ultimately responsible (and gets beaten up) if a change turns out to be bad. Please respect this and give the release engineer your full co-operation when it comes to the FreeBSD-STABLE branch. The management of FreeBSD-STABLE may frequently seem to be overly conservative to the casual observer, but also bear in mind the fact that conservatism is supposed to be the hallmark of FreeBSD-STABLE and different rules apply there than in FreeBSD-CURRENT. There is also really no point in having FreeBSD-CURRENT be a testing ground if changes are merged over to FreeBSD-STABLE immediately. Changes need a chance to be tested by the FreeBSD-CURRENT developers, so allow some time to elapse before merging unless the FreeBSD-STABLE fix is critical, time sensitive or so obvious as to make further testing unnecessary (spelling fixes to manual pages, obvious bug/typo fixes, etc.) In other words, apply common sense.

Changes to the security branches (for example, `releng/9.3`) must be approved by a member of the Security Officer Team <security-officer@FreeBSD.org>, or in some cases, by a member of the Release Engineering Team <re@FreeBSD.org>.

7. Do not fight in public with other committers; it looks bad.

   This project has a public image to uphold and that image is very important to all of us, especially if we are to continue to attract new members. There will be occasions when, despite everyone's very best attempts at self-control, tempers are lost and angry words are exchanged. The best thing that can be done in such cases is to minimize the effects of this until everyone has cooled back down. Do not air angry words in public and do not forward private correspondence or other private communications to public mailing lists, mail aliases, instant messaging channels or social media sites. What people say one-to-one is often much less sugar-coated than what they would say in public, and such communications therefore have no place there - they only serve to inflame an already bad situation. If the person sending a flame-o-gram at least had the grace to send it privately, then have the grace to keep it private yourself. If you feel you are being unfairly treated by another developer, and it is causing you anguish, bring the matter up with core rather than taking it public. Core will do its best to play peace makers and get things back to sanity. In cases where the dispute involves a change to the codebase and the participants do not appear to be reaching an amicable agreement, core may appoint a mutually-agreeable third party to resolve the dispute. All parties involved must then agree to be bound by the decision reached by this third party.

8. Respect all code freezes and read the `committers` and `developers` mailing list on a timely basis so you know when a code freeze is in effect.

   Committing unapproved changes during a code freeze is a really big mistake and committers are expected to keep up-to-date on what is going on before jumping in after a long absence and committing 10 megabytes worth of accumulated stuff. People who abuse this on a regular basis will have their commit privileges suspended until they get back from the FreeBSD Happy Reeducation Camp we run in Greenland.

9. When in doubt on any procedure, ask first!

   Many mistakes are made because someone is in a hurry and just assumes they know the right way of doing something. If you have not done it before, chances are good that you do not actually know the way we do things and really need to ask first or you are going to completely embarrass yourself in public. There is no shame in asking “how in the heck do I do this?” We already know you are an intelligent person; otherwise, you would not be a committer.

10. Test your changes before committing them.

    This may sound obvious, but if it really were so obvious then we probably would not see so many cases of people clearly not doing this. If your changes are to the kernel, make sure you can still compile both GENERIC and LINT. If your changes are anywhere else, make sure you can still make world. If your changes are to a branch, make sure your testing occurs with a machine which is running that code. If you have a change which also may break another architecture, be sure and test on all supported architectures. Please refer to the FreeBSD Internal Page for a list of available resources. As other architectures are added to the FreeBSD supported platforms list, the appropriate shared testing resources will be made available.

11. Do not commit to contributed software without *explicit* approval from the respective maintainers.

    Contributed software is anything under the `src/contrib`, `src/crypto`, or `src/sys/contrib` trees.

    The trees mentioned above are for contributed software usually imported onto a vendor branch. Committing something there may cause unnecessary headaches when importing newer versions of the software. As a general consider sending patches upstream to the vendor. Patches may be committed to FreeBSD first with permission of the maintainer.

    Reasons for modifying upstream software range from wanting strict control over a tightly coupled dependency to lack of portability in the canonical repository's distribution of their code. Regardless of the reason, effort to minimize the maintenance burden of fork is helpful to fellow maintainers. Avoid committing trivial or cosmetic changes to files since it makes every merge thereafter more difficult: such patches need to be manually re-verified every import.

    If a particular piece of software lacks a maintainer, you are encouraged to take up owership. If you are unsure of the current maintainership email FreeBSD architecture and design mailing list and ask.

18.2. Policy on Multiple Architectures

FreeBSD has added several new architecture ports during recent release cycles and is truly no longer an i386™ centric operating system. In an effort to make it easier to keep FreeBSD portable across the platforms we support, core has developed this mandate:

> Our 32-bit reference platform is i386, and our 64-bit reference platform is amd64. Major design work (including major API and ABI changes) must prove itself on at least one 32-bit and at least one 64-bit platform, preferably the primary reference platforms, before it may be committed to the source tree.

The i386 and amd64 platforms were chosen due to being more readily available to developers and as representatives of more diverse processor and system designs - big versus little endian, register file versus register stack, different DMA and cache implementations, hardware page tables versus software TLB management etc.

We will continue to re-evaluate this policy as cost and availability of the 64-bit platforms change.

Developers should also be aware of our Tier Policy for the long term support of hardware architectures. The rules here are intended to provide guidance during the development process, and are distinct from the requirements for features and architectures listed in that section. The Tier rules for feature support on architectures at release-time are more strict than the rules for changes during the development process.

18.3. Other Suggestions

When committing documentation changes, use a spell checker before committing. For all XML docs, verify that the formatting directives are correct by running `make lint` and textproc/igor.

For manual pages, run sysutils/manck and textproc/igor over the manual page to verify all of the cross references and file references are correct and that the man page has all of the appropriate `MLINK`s installed.

Do not mix style fixes with new functionality. A style fix is any change which does not modify the functionality of the code. Mixing the changes obfuscates the functionality change when asking for differences between revisions, which can hide any new bugs. Do not include whitespace changes with content changes in commits to `doc/` . The extra clutter in the diffs makes the translators' job much more difficult. Instead, make any style or whitespace changes in separate commits that are clearly labeled as such in the commit message.

18.4. Deprecating Features

When it is necessary to remove functionality from software in the base system, follow these guidelines whenever possible:

1. Mention is made in the manual page and possibly the release notes that the option, utility, or interface is deprecated. Use of the deprecated feature generates a warning.
2. The option, utility, or interface is preserved until the next major (point zero) release.
3. The option, utility, or interface is removed and no longer documented. It is now obsolete. It is also generally a good idea to note its removal in the release notes.

18.5. Privacy and Confidentiality

1. Most FreeBSD business is done in public.

   FreeBSD is an *open* project. Which means that not only can anyone use the source code, but that most of the development process is open to public scrutiny.

2. Certain sensitive matters must remain private or held under embargo.

   There unfortunately cannot be complete transparency. As a FreeBSD developer you will have a certain degree of privileged access to information. Consequently you are expected to respect certain requirements for confidentiality. Sometimes the need for confidentiality comes from external collaborators or has a specific time limit. Mostly though, it is a matter of not releasing private communications.

3. The Security Officer has sole control over the release of security advisories.

   Where there are security problems that affect many different operating systems, FreeBSD frequently depends on early access to be able to prepare advisories for coordinated release. Unless FreeBSD developers can be trusted to maintain security, such early access will not be made available. The Security Officer is responsible for controlling pre-release access to information about vulnerabilities, and for timing the release of all advisories. He may request help under condition of confidentiality from any developer with relevant knowledge to prepare security fixes.

4. Communications with Core are kept confidential for as long as necessary.

   Communications to core will initially be treated as confidential. Eventually however, most of Core's business will be summarized into the monthly or quarterly core reports. Care will be taken to avoid publicising any sensitive details. Records of some particularly sensitive subjects may not be reported on at all and will be retained only in Core's private archives.

5. Non-disclosure Agreements may be required for access to certain commercially sensitive data.

   Access to certain commercially sensitive data may only be available under a Non-Disclosure Agreement. The FreeBSD Foundation legal staff must be consulted before any binding agreements are entered into.

6. Private communications must not be made public without permission.

   Beyond the specific requirements above there is a general expectation not to publish private communications between developers without the consent of all parties involved. Ask permission before forwarding a message onto a public mailing list, or posting it to a forum or website that can be accessed by other than the original correspondents.

7. Communications on project-only or restricted access channels must be kept private.

   Similarly to personal communications, certain internal communications channels, including FreeBSD Committer only mailing lists and restricted access IRC channels are considered private communications. Permission is required to publish material from these sources.

8. Core may approve publication.

   Where it is impractical to obtain permission due to the number of correspondents or where permission to publish is unreasonably withheld, Core may approve release of such private matters that merit more general publication.

19. Support for Multiple Architectures

FreeBSD is a highly portable operating system intended to function on many different types of hardware architectures. Maintaining clean separation of Machine Dependent (MD) and Machine Independent (MI) code, as well as minimizing MD code, is an important part of our strategy to remain agile with regards to current hardware trends. Each new hardware architecture supported by FreeBSD adds substantially to the cost of code maintenance, toolchain support, and release engineering. It also dramatically increases the cost of effective testing of kernel changes. As such, there is strong motivation to differentiate between classes of support for various architectures while remaining strong in a few key architectures that are seen as the FreeBSD “target audience”.

19.1. Statement of General Intent

The FreeBSD Project targets "production quality commercial off-the-shelf (COTS) workstation, server, and high-end embedded systems". By retaining a focus on a narrow set of architectures of interest in these environments, the FreeBSD Project is able to maintain high levels of quality, stability, and performance, as well as minimize the load on various support teams on the project, such as the ports team, documentation team, security officer, and release engineering teams. Diversity in hardware support broadens the options for FreeBSD consumers by offering new features and usage opportunities, but these benefits must always be carefully considered in terms of the real-world maintenance cost associated with additional platform support.

The FreeBSD Project differentiates platform targets into four tiers. Each tier includes a list of guarantees consumers may rely on as well as obligations by the Project and developers to fulfill those guarantees. These lists define the minimum guarantees for each tier. The Project and developers may provide additional levels of support beyond the minimum guarantees for a given tier, but such additional support is not guaranteed. Each platform target is assigned to a specific tier for each stable branch. As a result, a platform target might be assigned to different tiers on concurrent stable branches.

19.2. Platform Targets

Support for a hardware platform consists of two components: kernel support and userland Application Binary Interfaces (ABIs). Kernel platform support includes things needed to run a FreeBSD kernel on a hardware platform such as machine-dependent virtual memory management and device drivers. A userland ABI specifies an interface for user processes to interact with a FreeBSD kernel and base system libraries. A userland ABI includes system call interfaces, the layout and semantics of public data structures, and the layout and semantics of arguments passed to subroutines. Some components of an ABI may be defined by specifications such as the layout of C++ exception objects or calling conventions for C functions.

A FreeBSD kernel also uses an ABI (sometimes referred to as the Kernel Binary Interface (KBI)) which includes the semantics and layouts of public data structures and the layout and semantics of arguments to public functions within the kernel itself.

A FreeBSD kernel may support multiple userland ABIs. For example, FreeBSD's amd64 kernel supports FreeBSD amd64 and i386 userland ABIs as well as Linux x86_64 and i386 userland ABIs. A FreeBSD kernel should support a “native” ABI as the default ABI. The native “ABI” generally shares certain properties with the kernel ABI such as the C calling convention, sizes of basic types, etc.

Tiers are defined for both kernels and userland ABIs. In the common case, a platform's kernel and FreeBSD ABIs are assigned to the same tier.

19.3. Tier 1: Fully-Supported Architectures

Tier 1 platforms are the most mature FreeBSD platforms. They are supported by the security officer, release engineering, and port management teams. Tier 1 architectures are expected to be Production Quality with respect to all aspects of the FreeBSD operating system, including installation and development environments.

The FreeBSD Project provides the following guarantees to consumers of Tier 1 platforms:

- Official FreeBSD release images will be provided by the release engineering team.

- Binary updates and source patches for Security Advisories and Errata Notices will be provided for supported releases.
- Source patches for Security Advisories will be provided for supported branches.
- Binary updates and source patches for cross-platform Security Advisories will typically be provided at the time of the announcement.
- Changes to userland ABIs will generally include compatibility shims to ensure correct operation of binaries compiled against any stable branch where the platform is Tier 1. These shims might not be enabled in the default install. If compatibility shims are not provided for an ABI change, the lack of shims will be clearly documented in the release notes.
- Changes to certain portions of the kernel ABI will include compatibility shims to ensure correct operation of kernel modules compiled against the oldest supported release on the branch. Note that not all parts of the kernel ABI are protected.
- Official binary packages for third party software will be provided by the ports team. For embedded architectures, these packages may be cross-built from a different architecture.
- Most relevant ports should either build or have the appropriate filters to prevent inappropriate ones from building.
- New features which are not inherently platform-specific will be fully functional on all Tier 1 architectures.
- Features and compatibility shims used by binaries compiled against older stable branches may be removed in newer major versions. Such removals will be clearly documented in the release notes.
- Tier 1 platforms should be fully documented. Basic operations will be documented in the FreeBSD Handbook.
- Tier 1 platforms will be included in the source tree.
- Tier 1 platforms should be self-hosting either via the in-tree toolchain or an external toolchain. If an external toolchain is required, official binary packages for an external toolchain will be provided.

To maintain maturity of Tier 1 platforms, the FreeBSD Project will maintain the following resources to support development:

- Build and test automation support either in the FreeBSD.org cluster or some other location easily available for all developers. Embedded platforms may substitute an emulator available in the FreeBSD.org cluster for actual hardware.
- Inclusion in the **`make universe`** and **`make tinderbox`** targets.
- Dedicated hardware in one of the FreeBSD clusters for package building (either natively or via qemu-user).

Collectively, developers are required to provide the following to maintain the Tier 1 status of a platform:

- Changes to the source tree should not knowingly break the build of a Tier 1 platform.
- Tier 1 architectures must have a mature, healthy ecosystem of users and active developers.
- Developers should be able to build packages on commonly available, non-embedded Tier 1 systems. This can mean either native builds if non-embedded systems are commonly available for the platform in question, or it can mean cross-builds hosted on some other Tier 1 architecture.
- Changes cannot break the userland ABI. If an ABI change is required, ABI compatibility for existing binaries should be provided via use of symbol versioning or shared library version bumps.
- Changes merged to stable branches cannot break the protected portions of the kernel ABI. If a kernel ABI change is required, the change should be modified to preserve functionality of existing kernel modules.

19.4. Tier 2: Developmental and Niche Architectures

Tier 2 platforms are functional, but less mature FreeBSD platforms. They are not supported by the security officer, release engineering, and port management teams.

Tier 2 platforms may be Tier 1 platform candidates that are still under active development. Architectures reaching end of life may also be moved from Tier 1 status to Tier 2 status as the availability of resources to continue to maintain the system in a Production Quality state diminishes. Well-supported niche architectures may also be Tier 2.

The FreeBSD Project provides the following guarantees to consumers of Tier 2 platforms:

- The ports infrastructure should include basic support for Tier 2 architectures sufficient to support building ports and packages. This includes support for basic packages such as ports-mgmt/pkg, but there is no guarantee that arbitrary ports will be buildable or functional.
- New features which are not inherently platform-specific should be feasible on all Tier 2 architectures if not implemented.
- Tier 2 platforms will be included in the source tree.
- Tier 2 platforms should be self-hosting either via the in-tree toolchain or an external toolchain. If an external toolchain is required, official binary packages for an external toolchain will be provided.
- Tier 2 platforms should provide functional kernels and userlands even if an official release distribution is not provided.

To maintain maturity of Tier 2 platforms, the FreeBSD Project will maintain the following resources to support development:

- Inclusion in the **`make universe`** and **`make tinderbox`** targets.

Collectively, developers are required to provide the following to maintain the Tier 2 status of a platform:

- Changes to the source tree should not knowingly break the build of a Tier 2 platform.
- Tier 2 architectures must have an active ecosystem of users and developers.
- While changes are permitted to break the userland ABI, the ABI should not be broken gratuitously. Significant userland ABI changes should be restricted to major versions.
- New features that are not yet implemented on Tier 2 architectures should provide a means of disabling them on those architectures.

19.5. Tier 3: Experimental Architectures

Tier 3 platforms have at least partial FreeBSD support. They are *not* supported by the security officer, release engineering, and port management teams.

Tier 3 platforms are architectures in the early stages of development, for non-mainstream hardware platforms, or which are considered legacy systems unlikely to see broad future use. Initial support for Tier 3 platforms may exist in a separate repository rather than the main source repository.

The FreeBSD Project provides no guarantees to consumers of Tier 3 platforms and is not committed to maintaining resources to support development. Tier 3 platforms may not always be buildable, nor are any kernel or userland ABIs considered stable.

19.6. Tier 4: Unsupported Architectures

Tier 4 platforms are not supported in any form by the project.

All systems not otherwise classified are Tier 4 systems. When a platform transitions to Tier 4, all support for the platform is removed from the source and ports trees. Note that ports support should remain as long as the platform is supported in a branch supported by ports.

19.7. Policy on Changing the Tier of an Architecture

Systems may only be moved from one tier to another by approval of the FreeBSD Core Team, which shall make that decision in collaboration with the Security Officer, Release Engineering, and ports management teams. For a platform to be promoted to a higher tier, any missing support guarantees must be satisfied before the promotion is completed.

20. Ports Specific FAQ

20.1. Adding a New Port

Q: How do I add a new port?

A: First, please read the section about repository copies.

The easiest way to add a new port is the `addport` script located in the `ports/Tools/scripts` directory. It adds a port from the directory specified, determining the category automatically from the port `Makefile`. It also adds an entry to the port's category `Makefile`. It was written by Michael Haro <mharo@FreeBSD.org>, Will Andrews <will@FreeBSD.org>, and Renato Botelho <garga@FreeBSD.org>. When sending questions about this script to the FreeBSD ports mailing list, please also CC Chris Rees <crees@FreeBSD.org>, the current maintainer.

Q: Any other things I need to know when I add a new port?

A: Check the port, preferably to make sure it compiles and packages correctly. This is the recommended sequence:

```
# make install
# make package
# make deinstall
# pkg add package you built above
# make deinstall
# make reinstall
# make package
```

The Porters Handbook contains more detailed instructions.

Use portlint(1) to check the syntax of the port. You do not necessarily have to eliminate all warnings but make sure you have fixed the simple ones.

If the port came from a submitter who has not contributed to the Project before, add that person's name to the Additional Contributors section of the FreeBSD Contributors List.

Close the PR if the port came in as a PR. To close a PR, change the state to `Issue Resolved` and the resolution as `Fixed`.

20.2. Removing an Existing Port

Q: How do I remove an existing port?

A: First, please read the section about repository copies. Before you remove the port, you have to verify there are no other ports depending on it.

- Make sure there is no dependency on the port in the ports collection:
  - The port's PKGNAME appears in exactly one line in a recent INDEX file.

- No other ports contains any reference to the port's directory or PKGNAME in their Makefiles

Tip

When using Git, consider using `git grep`, it is much faster than `grep -r`.

- Then, remove the port:

  1. Remove the port's files and directory with `svn remove`.
  2. Remove the `SUBDIR` listing of the port in the parent directory `Makefile`.
  3. Add an entry to `ports/MOVED` .
  4. Search for entries in `ports/security/vuxml/vuln.xml` and adjust them accordingly. In particular, check for previous packages with the new name which version could include the new port.
  5. Remove the port from `ports/LEGAL` if it is there.

Alternatively, you can use the `rmport` script, from `ports/Tools/scripts` . This script was written by Vasil Dimov <vd@FreeBSD.org>. When sending questions about this script to the FreeBSD ports mailing list, please also CC Chris Rees <crees@FreeBSD.org>, the current maintainer.

20.3. Re-adding a Deleted Port

Q: How do I re-add a deleted port?

A: This is essentially the reverse of deleting a port.

Important

Do not use `svn add` to add the port. Follow these steps. If they are unclear, or are not working, ask for help, do not just `svn add` the port.

1. Figure out when the port was removed. Use this list, or look for the port on freshports, and then copy the last living revision of the port:

```
% cd /usr/ports/ category
% svn cp 'svn+ssh://repo.freebsd.org/ports/head/  category /portname /
@XXXXXX' portname
```

   Pick the revision that is just before the removal. For example, if the revision where it was removed is `269874`, use `269873`.

   It is also possible to specify a date. In that case, pick a date that is before the removal but after the last commit to the port.

```
% cd /usr/ports/ category
% svn cp 'svn+ssh://repo.freebsd.org/ports/head/  category /portname /@{YYYY-MM-
DD}' portname
```

2. Make the changes necessary to get the port working again. If it was deleted because the distfiles are no longer available, either volunteer to host the distfiles, or find someone else to do so.

3. If some files have been added, or were removed during the resurrection process, use `svn add` or `svn remove` to make sure all the files in the port will be committed.
4. Restore the `SUBDIR` listing of the port in the parent directory `Makefile`, keeping the entries sorted.
5. Delete the port entry from `ports/MOVED` .
6. If the port had an entry in `ports/LEGAL` , restore it.
7. `svn commit` these changes, preferably in one step.

Tip

The `addport` script mentioned in Q & A 20.1, "Adding a New Port" now detects when the port to add has previously existed, and attempts to handle all except the `ports/LEGAL` step automatically.

20.4. Repository Copies

Q: When do we need a repository copy?

A: When you want to add a port that is related to any port that is already in the tree in a separate directory, you have to do a repository copy. Here *related* means it is a different version or a slightly modified version. Examples are `print/ghostscript*` (different versions) and `x11-wm/windowmaker*` (English-only and internationalized version).

Another example is when a port is moved from one subdirectory to another, or when the name of a directory must be changed because the authors renamed their software even though it is a descendant of a port already in a tree.

Q: What do I need to do?

A: With Subversion, a repo copy can be done by any committer:

- Doing a repo copy:
  1. Verify that the target directory does not exist.
  2. Use `svn up` to make certain the original files, directories, and checkout information is current.
  3. Use `svn move` or `svn copy` to do the repo copy.
  4. Upgrade the copied port to the new version. Remember to add or change the `PKGNAMEPREFIX` or `PKGNAMESUFFIX` so there are no duplicate ports with the same name. In some rare cases it may be necessary to change the `PORTNAME` instead of adding `PKGNAMEPREFIX` or `PKGNAMESUFFIX`, but this is only done when it is really needed — for example, using an existing port as the base for a very similar program with a different name, or upgrading a port to a new upstream version which actually changes the distribution name, like the transition from `textproc/libxml` to `textproc/libxml2` . In most cases, adding or changing `PKGNAMEPREFIX` or `PKGNAMESUFFIX` suffices.
  5. Add the new subdirectory to the `SUBDIR` listing in the parent directory `Makefile`. You can run `make checksubdirs` in the parent directory to check this.
  6. If the port changed categories, modify the `CATEGORIES` line of the port's `Makefile` accordingly
  7. Add an entry to `ports/MOVED` , if you remove the original port.

8. Commit all changes on one commit.

- When removing a port:

1. Perform a thorough check of the ports collection for any dependencies on the old port location/name, and update them. Running `grep` on `INDEX` is not enough because some ports have dependencies enabled by compile-time options. A full `grep -r` of the ports collection is recommended.

2. Remove the old port and the old `SUBDIR` entry.

3. Add an entry to `ports/MOVED` .

- After repo moves ("rename" operations where a port is copied and the old location is removed):

  - Follow the same steps that are outlined in the previous two entries, to activate the new location of the port and remove the old one.

20.5. Ports Freeze

Q: What is a "ports freeze"?

A: A "ports freeze" was a restricted state the ports tree was put in before a release. It was used to ensure a higher quality for the packages shipped with a release. It usually lasted a couple of weeks. During that time, build problems were fixed, and the release packages were built. This practice is no longer used, as the packages for the releases are built from the current stable, quarterly branch.

For more information on how to merge commits to the quarterly branch, see Q:.

20.6. Quarterly Branches

Q: What is the procedure to request authorization for merging a commit to the quarterly branch?

A: When doing the commit, add the branch name to the `MFH:` line, for example:

```
MFH: 2014Q1
```

It will automatically notify the Ports Security Team <ports-secteam@FreeBSD.org> and the Ports Management Team <portmgr@FreeBSD.org>. They will then decide if the commit can be merged and answer with the procedure.

If the commit has already been made, send an email to the Ports Security Team <ports-secteam@FreeBSD.org> and the Ports Management Team <portmgr@FreeBSD.org> with the revision number and a small description of why the commit needs to be merged.

Tip

If the MFH is covered by a blanket approval, please explain why with a couple of words on the `MFH` line, so that the reviewing team can skip this commit and save time. For example:

```
MFH:  2014Q1 (runtime fix)
MFH:  2014Q1 (browser blanket)
```

The list of blanket approvals is available in Q:.

Q: Are there any changes that can be merged without asking for approval?

A: The following blanket approvals for merging to the quarterly branches are in effect:

Note

This blanket approval also applies to direct commits for ports that have been removed from `head`.

Important

These fixes *must* be tested on the quarterly branch.

- Fixes that do not result in a change in contents of the resulting package. For example:
  - `pkg-descr:WWW:` URL updates (existing 404, moved or incorrect)
- Build, runtime or packaging fixes, if the quarterly branch version is currently broken.
- Missing dependencies (detected, linked against but not registered via `*_DEPENDS`).
- Fixing shebangs, stripping installed libraries and binaries, and plist fixes.
- Backport of security and reliability fixes which only result in `PORTREVISION` bumps and no changes to enabled features. for example, adding a patch fixing a buffer overflow.
- Minor version changes that do nothing but fix security or crash-related issues.
- Adding/fixing `CONFLICTS`.
- Web Browsers, browser plugins, and their required dependencies.

Important

Commits that are not covered by these blanket approvals always require explicit approval of either Ports Security Team <ports-secteam@FreeBSD.org> or Ports Management Team <portmgr@FreeBSD.org>.

Q: What is the procedure for merging commits to the quarterly branch?

A: A script is provided to automate merging a specific commit: `ports/Tools/scripts/mfh` . It is used as follows:

```
% /usr/ports/Tools/scripts/mfh 380362
 U   2015Q1
Checked out revision 380443.
A    2015Q1/security
Updating '2015Q1/security/rubygem-sshkit':
A    2015Q1/security/rubygem-sshkit
A    2015Q1/security/rubygem-sshkit/Makefile
A    2015Q1/security/rubygem-sshkit/distinfo
A    2015Q1/security/rubygem-sshkit/pkg-descr
Updated to revision 380443.
--- Merging r380362 into '2015Q1':
```

```
U    2015Q1/security/rubygem-sshkit/Makefile
U    2015Q1/security/rubygem-sshkit/distinfo
--- Recording mergeinfo for merge of r380362 into '2015Q1':
 U   2015Q1
--- Recording mergeinfo for merge of r380362 into '2015Q1/security':
 G   2015Q1/security
--- Eliding mergeinfo from '2015Q1/security':
 U   2015Q1/security
--- Recording mergeinfo for merge of r380362 into '2015Q1/security/rubygem-sshkit':
 G   2015Q1/security/rubygem-sshkit
--- Eliding mergeinfo from '2015Q1/security/rubygem-sshkit':
 U   2015Q1/security/rubygem-sshkit
 M      2015Q1
M       2015Q1/security/rubygem-sshkit/Makefile
M       2015Q1/security/rubygem-sshkit/distinfo
Index: 2015Q1/security/rubygem-sshkit/Makefile
===================================================================
--- 2015Q1/security/rubygem-sshkit/Makefile     (revision 380443)
+++ 2015Q1/security/rubygem-sshkit/Makefile     (working copy)
@@ -2,7 +2,7 @@
 # $FreeBSD: head/en_US.ISO8859-1/articles/committers-guide/article.xml 53619 ↺
2019-11-21 17:38:49Z jhb $

 PORTNAME=      sshkit
-PORTVERSION=   1.6.1
+PORTVERSION=   1.7.0
 CATEGORIES=    security rubygems
 MASTER_SITES=  RG

Index: 2015Q1/security/rubygem-sshkit/distinfo
===================================================================
--- 2015Q1/security/rubygem-sshkit/distinfo     (revision 380443)
+++ 2015Q1/security/rubygem-sshkit/distinfo     (working copy)
@@ -1,2 +1,2 @@
-SHA256 (rubygem/sshkit-1.6.1.gem) = ↺
8ca67e46bb4ea50fdb0553cda77552f3e41b17a5aa919877d93875dfa22c03a7
-SIZE (rubygem/sshkit-1.6.1.gem) = 135680
+SHA256 (rubygem/sshkit-1.7.0.gem) = ↺
90effd1813363bae7355f4a45ebc8335a8ca74acc8d0933ba6ee6d40f281a2cf
+SIZE (rubygem/sshkit-1.7.0.gem) = 136192
Index: 2015Q1
===================================================================
--- 2015Q1      (revision 380443)
+++ 2015Q1      (working copy)

Property changes on: 2015Q1
___________________________________________________________________
Modified: svn:mergeinfo
   Merged /head:r380362
Do you want to commit? (no = start a shell) [y/n]
```

At that point, the script will either open a shell for you to fix things, or open your text editor with the commit message all prepared and then commit the merge.

The script assumes that you can connect to `repo.FreeBSD.org` with SSH directly, so if your local login name is different than your FreeBSD cluster account, you need a few lines in your `~/.ssh/config`:

```
Host *.freebsd.org
    User freebsd-login
```

Tip

The script is also able to merge more than one revision at a time. If there have been other updates to the port since the branch was created that have not been merged because they were not security related. Add the different revisions *in the order they were committed* on the `mfh` line. The new commit log message will contain the combined log messages from all the original commits. These messages *must* be edited to show what is actually being done with the new commit.

```
% /usr/ports/Tools/scripts/mfh r407208 r407713 r407722 r408567 ↺
r408943 r410728
```

Note

The mfh script can also take an optional first argument, the branch where the merge is being done. Only the latest quarterly branch is supported, so specifying the branch is discouraged. To be safe, the script will give a warning if the quarterly branch is not the latest:

```
% /usr/ports/Tools/scripts/mfh 2016Q1 r407208 r407713
/!\ The latest branch is 2016Q2, do you really want to commit to ↺
2016Q1? [y/n]
```

20.7. Creating a New Category

Q: What is the procedure for creating a new category?

A: Please see Proposing a New Category in the Porter's Handbook. Once that procedure has been followed and the PR has been assigned to the Ports Management Team <portmgr@FreeBSD.org>, it is their decision whether or not to approve it. If they do, it is their responsibility to:

1. Perform any needed moves. (This only applies to physical categories.)
2. Update the `VALID_CATEGORIES` definition in `ports/Mk/bsd.port.mk` .
3. Assign the PR back to you.

Q: What do I need to do to implement a new physical category?

A: 1. Upgrade each moved port's `Makefile`. Do not connect the new category to the build yet.

   To do this, you will need to:

   1. Change the port's `CATEGORIES` (this was the point of the exercise, remember?) The new category is listed *first*. This will help to ensure that the `PKGORIGIN` is correct.
   2. Run a `make describe`. Since the top-level `make index` that you will be running in a few steps is an iteration of `make describe` over the entire ports hierarchy, catching any errors here will save you having to re-run that step later on.
   3. If you want to be really thorough, now might be a good time to run portlint(1).

2. Check that the `PKGORIGIN` s are correct. The ports system uses each port's `CATEGORIES` entry to create its `PKGORIGIN` , which is used to connect installed packages to the port directory they were built from. If this entry is wrong, common port tools like pkg_version(1) and portupgrade(1) fail.

   To do this, use the `chkorigin.sh` tool: `env PORTSDIR=/path/to/ports sh -e /path/to/ports/Tools/scripts/chkorigin.sh` . This will check *every* port in the ports tree, even those not connected to the build, so you can run it directly after the move operation. Hint: do not forget to look at the `PKGORIGIN` s of any slave ports of the ports you just moved!

3. On your own local system, test the proposed changes: first, comment out the `SUBDIR` entries in the old ports' categories' `Makefiles`; then enable building the new category in `ports/Makefile` . Run `make checksubdirs` in the affected category directories to check the `SUBDIR` entries. Next, in the `ports/` directory, run `make index` . This can take over 40 minutes on even modern systems; however, it is a necessary step to prevent problems for other people.

4. Once this is done, you can commit the updated `ports/Makefile` to connect the new category to the build and also commit the `Makefile` changes for the old category or categories.

5. Add appropriate entries to `ports/MOVED` .

6. Update the documentation by modifying:

   - the list of categories in the Porter's Handbook

   - `doc/en_US.ISO8859-1/htdocs/ports` . Note that these are now displayed by sub-groups, as specified in `doc/en_US.ISO8859-1/htdocs/ports/categories.descriptions` .

   (Note: these are in the docs, not the ports, repository). If you are not a docs committer, you will need to submit a PR for this.

7. Only once all the above have been done, and no one is any longer reporting problems with the new ports, should the old ports be deleted from their previous locations in the repository.

It is not necessary to manually update the ports web pages to reflect the new category. This is done automatically via the change to `en_US.ISO8859-1/htdocs/ports/categories` and the automated rebuild of `INDEX`.

Q: What do I need to do to implement a new virtual category?

A: This is much simpler than a physical category. Only a few modifications are needed:

- the list of categories in the Porter's Handbook

- `en_US.ISO8859-1/htdocs/ports/categories`

20.8. Miscellaneous Questions

Q: Are there changes that can be committed without asking the maintainer for approval?

A: Blanket approval for most ports applies to these types of fixes:

- Most infrastructure changes to a port (that is, modernizing, but not changing the functionality). For example, the blanket covers converting to new `USES` macros, enabling verbose builds, and switching to new ports system syntaxes.

- Trivial and *tested* build and runtime fixes.

Important

Exceptions to this are anything maintained by the Ports Management Team <portmgr@FreeBSD.org>, or the Security Officer Team <security-officer@FreeBSD.org>. No unauthorized commits may ever be made to ports maintained by those groups.

Q: How do I know if my port is building correctly or not?

A: The packages are built multiple times each week. If a port fails, the maintainer will receive an email from `pkg-fallout@FreeBSD.org`.

Reports for all the package builds (official, experimental, and non-regression) are aggregated at pkg-status.FreeBSD.org.

Q: I added a new port. Do I need to add it to the `INDEX`?

A: No. The file can either be generated by running `make index`, or a pre-generated version can be downloaded with `make fetchindex`.

Q: Are there any other files I am not allowed to touch?

A: Any file directly under `ports/`, or any file under a subdirectory that starts with an uppercase letter (`Mk/`, `Tools/`, etc.). In particular, the Ports Management Team <portmgr@FreeBSD.org> is very protective of `ports/Mk/bsd.port*.mk` so do not commit changes to those files unless you want to face their wrath.

Q: What is the proper procedure for updating the checksum for a port distfile when the file changes without a version change?

A: When the checksum for a distribution file is updated due to the author updating the file without changing the port revision, the commit message includes a summary of the relevant diffs between the original and new distfile to ensure that the distfile has not been corrupted or maliciously altered. If the current version of the port has been in the ports tree for a while, a copy of the old distfile will usually be available on the ftp servers; otherwise the author or maintainer should be contacted to find out why the distfile has changed.

Q: How can an experimental test build of the ports tree (*exp-run*) be requested?

A: An exp-run must be completed before patches with a significant ports impact are committed. The patch can be against the ports tree or the base system.

Full package builds will be done with the patches provided by the submitter, and the submitter is required to fix detected problems (*fallout*) before commit.

1. Go to the Bugzilla new PR page.
2. Select the product your patch is about.
3. Fill in the bug report as normal. Remember to attach the patch.
4. If at the top it says “Show Advanced Fields” click on it. It will now say “Hide Advanced Fields”. Many new fields will be available. If it already says “Hide Advanced Fields”, no need to do anything.
5. In the “Flags” section, set the “exp-run” one to `?`. As for all other fields, hovering the mouse over any field shows more details.
6. Submit. Wait for the build to run.

7. Ports Management Team <portmgr@FreeBSD.org> will replies with a possible fallout.

8. Depending on the fallout:

    - If there is no fallout, the procedure stops here, and the change can be committed, pending any other approval required.

    - a. If there is fallout, it *must* be fixed, either by fixing the ports directly in the ports tree, or adding to the submitted patch.

      b. When this is done, go back to step 6 saying the fallout was fixed and wait for the exp-run to be run again. Repeat as long as there are broken ports.

21. Issues Specific to Developers Who Are Not Committers

A few people who have access to the FreeBSD machines do not have commit bits. Almost all of this document will apply to these developers as well (except things specific to commits and the mailing list memberships that go with them). In particular, we recommend that you read:

- Administrative Details
- Conventions

Note

Get your mentor to add you to the "Additional Contributors" (`doc/en_US.ISO8859-1/articles/contributors/contrib.additional.xml`), if you are not already listed there.

- Developer Relations
- SSH Quick-Start Guide
- The FreeBSD Committers' Big List of Rules

22. Information About Google Analytics

As of December 12, 2012, Google Analytics was enabled on the FreeBSD Project website to collect anonymized usage statistics regarding usage of the site. The information collected is valuable to the FreeBSD Documentation Project, to identify various problems on the FreeBSD website.

22.1. Google Analytics General Policy

The FreeBSD Project takes visitor privacy very seriously. As such, the FreeBSD Project website honors the "Do Not Track" header *before* fetching the tracking code from Google. For more information, please see the FreeBSD Privacy Policy.

Google Analytics access is *not* arbitrarily allowed — access must be requested, voted on by the Documentation Engineering Team <doceng@FreeBSD.org>, and explicitly granted.

Requests for Google Analytics data must include a specific purpose. For example, a valid reason for requesting access would be "to see the most frequently used web browsers when viewing FreeBSD web pages to ensure page rendering speeds are acceptable."

Conversely, “to see what web browsers are most frequently used” (without stating *why*) would be rejected.

All requests must include the timeframe for which the data would be required. For example, it must be explicitly stated if the requested data would be needed for a timeframe covering a span of 3 weeks, or if the request would be one-time only.

Any request for Google Analytics data without a clear, reasonable reason beneficial to the FreeBSD Project will be rejected.

22.2. Data Available Through Google Analytics

A few examples of the types of Google Analytics data available include:

- Commonly used web browsers
- Page load times
- Site access by language

23. Miscellaneous Questions

Q: How do I add a new file to a branch?

A: To add a file onto a branch, simply checkout or update to the branch you want to add to and then add the file using the add operation as you normally would. This works fine for the `doc` and `ports` trees. The `src` tree uses SVN and requires more care because of the `mergeinfo` properties. See the Subversion Primer for details on how to perform an MFC.

Q: How do I access `people.FreeBSD.org` to put up personal or project information?

A: `people.FreeBSD.org` is the same as `freefall.FreeBSD.org`. Just create a `public_html` directory. Anything you place in that directory will automatically be visible under `https://people.FreeBSD.org/` .

Q: Where are the mailing list archives stored?

A: The mailing lists are archived under `/local/mail` on `freefall.FreeBSD.org`.

Q: I would like to mentor a new committer. What process do I need to follow?

A: See the New Account Creation Procedure document on the internal pages.

24. Benefits and Perks for FreeBSD Committers

24.1. Recognition

Recognition as a competent software engineer is the longest lasting value. In addition, getting a chance to work with some of the best people that every engineer would dream of meeting is a great perk!

24.2. FreeBSD Mall

FreeBSD committers can get a free 4-CD or DVD set at conferences from FreeBSD Mall, Inc..

24.3. IRC

In addition, developers may request a cloaked hostmask for their account on the Freenode IRC network in the form of `freebsd/developer/`*`freefall name`* or `freebsd/developer/`*`NickServ name`*`.` To request a cloak, send an email to <irc@FreeBSD.org> with your requested hostmask and NickServ account name.

24.4. `Gandi.net`

Gandi provides website hosting, cloud computing, domain registration, and X.509 certificate services.

Gandi offers an E-rate discount to all FreeBSD developers. Send mail to <non-profit@gandi.net> using your `@freebsd.org` mail address, and indicate your Gandi handle.