Updating FreeBSD From Git

BY DREW GURKOWSKI

With FreeBSD's ongoing migration to git from subversion, the system for updating FreeBSD from source has adapted. This guide will cover getting sources from git, updating them, and how to bisect those sources. It is meant as an introduction to the new mechanics for general users.

1. Keeping Current With FreeBSD src tree

To begin, the source tree must be downloaded. This can be done quite simply. First step: cloning a tree. This downloads the entire tree. There are two ways to download. By default, git will do a deep clone, which matches what most people want. However, there are times that you may wish to do a shallow clone.

1.1 Branch names

The branch names in the new git repository are similar to the subversion names. For the stable branches, they are `stable/X` where `X` is the major release number (like 12 or 13). The development branch for `-CURRENT` in the new repository is `main`.

Note: The main branch is the default branch if you omit the `-b branch` options below.

1.2 Repositories

- The official geographically distributed mirror for the general public is git.FreeBSD.org, The access URL is: https://git.freebsd.org/src.git
- The repository is also accessible by SSH: ssh://anongit@git.freebsd.org/src.git
- There are several officially maintained external mirrors. The list is available at https://docs.freebsd.org/en/books/handbook/mirrors/#external-mirrors
- For using web browser to view the content, there is a cgit web interface at https://cgit.freebsd.org/src
- There is an old experimental github repository at https://github.com/freebsd/freebsd-legacy/

(was https://github.com/freebsd/freebsd) similar to the new Git repository. However, there are a large number of mistakes in the github repository that required us to regenerate the export when we migrated to having a git repository be the source of truth for the project.

The hashes are different between them. For migrating from the old repository to the new one, please refer to https://github.com/freebsd/freebsd-legacy/commit/de1aa3dab-23c06fec962a14da3e7b4755c5880cf

Use the repository of choice in place of $URL in the following commands.

1.2.1 Install git From Ports/Pkg

Before cloning the tree, git will need to be installed. The simplest way of doing so is through packages.

```
# pkg install git
```

There are also git-lite and git-tiny, two packages with only essential dependencies available. They are sufficient for the commands in this article.

1.2.2 Deep Clone

A deep clone pulls in the entire tree, as well as all the history and branches. It's the easiest to do. It also allows you to use git's worktree feature to have all your active branches checked out into separate directories but with only one copy of the repository.

```
% git clone $URL -b branch [dir]
```

is how you make a deep clone. `branch` should be one of the branches listed in the **section 1.1**, if omitted, it will be the depository's default: main. Dir is an optional directory to place it in (the default will be the name of the repository you are clone (src). For example:

```
% git clone https://git.freebsd.org/src.git
```

You'll want a deep clone if you are interested in the history, plan on making local changes, or plan on working on more than one branch. It's the easiest to keep up to date as well. If you are interested in the history, but are working with only one branch and are short on space, you can also use `--single-branch` to only download the one branch (though some merge commits will not reference the merged-from branch which may be important for some users who are interested in detailed versions of history).

1.2.3 Shallow Clone

A shallow clone copies just the most current code, but none or little of the history. This can be useful when you need to build a specific revision of FreeBSD, or when you are just starting out and plan to track the tree more fully. You can also use it to limit history to only so many revisions.

```
% git clone -b branch --depth 1 $URL [dir]
```

An example using the default branch:

```
% git clone --depth 1 https://git.freebsd.org/src.git
```

This clones the repository, but only has the most recent revision in the repository. The rest of the history is not downloaded. Should you change your mind later, you can do git fetch --unshallow to get the complete history.

2. Building and Updating from Source

After cloning the FreeBSD repository, the next step is to build from source. The process of building remains relatively unchanged, using make and install. Git and offer git pull and git checkout for updating and selecting specific branches or revisions.

2.1 Building From Source

Building can be done as described in the handbook [3], eg:

```
% cd src
% make buildworld
% make buildkernel
% make installkernel
% make installworld
```

Note that you can specify `-j` to make to speed up with parallelism.

2.2 Updating From Source

The following command will update both types of trees. This will pull all revisions since the last update.

```
% git pull --ff-only
```

This will update the tree. In git, a fast forward merge is one that only needs to set a new branch pointer and doesn't need to re-create the commits. By always doing a fast forward merge/pull, you'll ensure that you have an identical copy of the FreeBSD tree. This will be important if you want to maintain local patches.

See below for how to manage local changes. The simplest is to use:

```
% git pull --autostash
```

but more sophisticated options are available.

2.3 Selecting a Specific Revision

In git, the **`git checkout`** command can be used to checkout a specific revision as well as branches. Git's revisions are the long hashes rather than a sequential number.

When you checkout a specific revision, just specify the hash you want on the command line (the git log command can help you decide which hash you might want):

```
% git checkout 08b8197a74
```

However, as with many things git, it's not so simple. You'll be greeted with a message similar to the following:

```
Note: checking out '08b8197a742a96964d2924391bf9fdfeb788865d'.

You are in 'detached HEAD' state. You can look around, make experimental
changes and commit them, and you can discard any commits you make in this
state without impacting any branches by performing another checkout.

If you want to create a new branch to retain commits you create, you may
do so (now or later) by using -b with the checkout command again. Example:

  git checkout -b <new-branch-name>

HEAD is now at 08b8197a742a hook gpiokeys.4 to the build
```

where the last line is generated from the hash you are checking out and the first line of the commit message from that revision. Hashes can also be abbreviated. That's why you'll see them have different lengths in different commands or their outputs. These super long hashes are often unique after some characters, depends on the size of the repository so git lets you abbreviate and is somewhat inconsistent about how it presents them to users. `git rev-parse --short <full-hash>` will show the short hash which has the enough length to distinguish in the repository. The current length of FreeBSD src repository is 12.

3. Bisecting/Other Considerations

3.1 Bisecting With git bisect

Sometimes, things go wrong. The last revision worked, but the one you just updated to does not. A developer may ask to bisect the problem to track down which commit caused the regression.

If you've read the last section, you may be thinking to yourself "How the heck do I bisect with crazy revision numbers like that?" then this section is for you. It's also for you if you didn't think that, but also want to bisect.

Fortunately, git offers the `git bisect` command. Here's a brief outline in how to use it. For more information, I'd suggest https://git-scm.com/docs/git-bisect for more details. The man page is good at describing what can go wrong, what to do when revisions won't build, when you want to use terms other than good and bad, etc, none of which will be covered here.

`git bisect start` will start the bisection process. Next, you need to tell a range to go through. `git bisect good XXXXXX` will tell it the working revision and `git bisect bad XXXXX` will tell it the bad revision. The bad revision will almost always be `HEAD` (a special tag for what you have checked out). The good revision will be the last one you checked out.

A quick aside: if you want to know the last revision you checked out, you should use `git reflog`:

```
5ef0bd68b515 (HEAD -> master, origin/master, origin/HEAD) HEAD@{0}: pull --ff-only: Fast-forward
a8163e165c5b (upstream/master) HEAD@{1}: checkout: moving from b6fb97efb682994f59b21fe4efb3fcfc0e5b9eeb to master
...
```

shows me moving the working tree to the master branch (a816…) and then updating from upstream (to 5ef0…). In this case, `bad` would be `HEAD` (or 5rf0bd68) and `good` would be a8163e165. As you can see from the output, HEAD@{1} also often works, but isn't foolproof if you've done other things to your git tree after updating, but before you discover the need to bisect.

Back to git bisect. Set the `good` revision first, then set the `bad` (though the order doesn't matter). When you set the bad revision, it will give you some statistics on the process:

```
% git bisect start
% git bisect good a8163e165c5b
% git bisect bad HEAD
```

```
Bisecting: 1722 revisions left to test after this (roughly 11 steps)
[c427b3158fd8225f6afc09e7e6f62326f9e4de7e] Fixup r361997 by balancing parens.
```

You'd then build/install that revision. If it's good you'd type `git bisect good` otherwise `git bisect bad`. You'll get a similar message to the above each step. When you are done, report the bad revision to the developer (or fix the bug yourself and send a patch). `git bisect reset` will end the process and return you back to where you started (usually tip of main). Again, the git-bisect manual (linked above) is a good resource for when things go wrong or for unusual cases.

3.2 Documents and Ports Considerations

The doc tree is the first repository converted to git. There is only one development branch in the repository, main, which contains the source of https://www.freebsd.org and https://docs.freebsd.org.

- The repository URL is at https://git.freebsd.org/doc.git
- The repository is also accessible with SSH: ssh://anongit@git.freebsd.org/doc.git
- And cgit web repository browser is at: https://cgit.freebsd.org/doc/

The ports tree operates a similar way. The branch names are different and the repos are in different locations.

- The repository URL is at https://git.freebsd.org/ports.git
- The repository is also accessible with SSH: ssh://anongit@git.freebsd.org/ports.git
- And cgit web repository browser is at: https://cgit.freebsd.org/ports/

As with ports, the latest development branch is main. The quarterly branches are named the same as in FreeBSD's svn repo. They are used by the latest and quarterly branches of the pkg.

3.3 Coping with Local Changes

Here's a small collection of topics that are more advanced for the user tracking FreeBSD. If you have no local changes, you can stop reading now (it's the last section and OK to skip).

One item that's important for all of them: all changes are local until pushed. Unlike svn, git uses a distributed model. For users, for most things, there's very little difference. However, if you have local changes, you can use the same tool to manage them as you use to pull in changes from FreeBSD. All changes that you've not pushed are local and can easily be modified (git rebase, discussed below does this).

3.4 Keeping local changes

The simplest way to keep local changes (especially trivial ones) is to use `git stash`. In its simplest form, you use `git stash` to record the changes (which pushes them onto the stash stack). Most people use this to save changes before updating the tree as described above. They then use `git stash apply` to re-apply them to the tree. The stash is a stack of changes that can be examined with `git stash list`. The git-stash man page (https://git-scm.com/docs/git-stash) has all the details.

This method is suitable when you have tiny tweaks to the tree. When you have anything non trivial, you'll likely be better off keeping a local branch and rebasing. It is also integrated with the `git pull` command: just add `--autostash` to the command line.

4. FreeBSD Branches

4.1 Keeping a local branch

It's much easier to keep a local branch with git than subversion. In subversion you need to merge the commit, and resolve the conflicts. This is manageable, but can lead to a convoluted history that's hard to upstream should that ever be necessary, or hard to replicate if you need to do so. Git also allows one to merge, along with the same problems. That's one way to manage the branch, but it's the least flexible.

Git has a concept of 'rebasing' which you can use to avoid these issues. The `git rebase` command will basically replay all the commits relative to the parent branch at a newer location on that parent branch. This section will briefly cover how to do this, but will not cover all scenarios.

4.2 Create a branch

Let's say you want to make a hack to FreeBSD's `ls(1)` command to never, ever do color. There's many reasons to do this, but this example will use that as a baseline. The FreeBSD `ls(1)` command changes from time to time, and you'll need to cope with those changes. Fortunately, with git rebase it usually is automatic.

```
% cd src
% git checkout main
% git checkout -b no-color-ls
% cd bin/ls
% vi ls.c # hack the changes in
% git diff # check the changes
```

```
diff --git a/bin/ls/ls.c b/bin/ls/ls.c
index 7378268867ef..cfc3f4342531 100644
--- a/bin/ls/ls.c
+++ b/bin/ls/ls.c
@@ -66,6 +66,7 @@ __FBSDID("$FreeBSD$");
 #include <stdlib.h>
 #include <string.h>
 #include <unistd.h>
+#undef COLORLS
 #ifdef COLORLS
 #include <termcap.h>
 #include <signal.h>
```

```
% # these look good, make the commit...
% git commit ls.c
```

The commit will pop you into an editor to describe what you've done. Once you enter that, you have your own `local` branch in the git repo. Build and install it like you normally would, following the directions in the handbook. git differs from other revision control systems in that you have to tell it explicitly which files to use. Here it's done on the commit command line, but you can also do it with `git add` which many of the more in depth tutorials cover.

5. Updating to New FreeBSD Releases

5.1 Updating to a New FreeBSD Revision

When it's time to bring in a new revision, it's almost the same as w/o the branches. You would update like you would above, but there's one extra command before you update, and one after. The following assumes you are starting with an unmodified tree. It's important to start rebasing operations with a clean tree (git usually requires this).

```
% git checkout main
% git pull
% git rebase -i main no-color-ls
```

This will bring up an editor that lists all the commits in it. For this example, don't change it at all. This is typically what you are doing while updating the baseline (though you also use the `git rebase` command to curate the commits you have in the branch).

Once you're done with the above, you've moved the commits to ls.c forward from the old revision of FreeBSD to the newer one.

Sometimes there's merge conflicts. That's OK. Don't panic. You'd handle them the same as you would any other merge conflicts. To keep it simple, I'll just describe a common issue you might see. A pointer to a more complete treatment can be found at the end of this section.

Let's say this includes changes upstream in a radical shift to terminfo as well as a name change for the option. When you updated, you might see something like this:

```
Auto-merging bin/ls/ls.c
CONFLICT (content): Merge conflict in bin/ls/ls.c
error: could not apply 646e0f9cda11... no color ls
Resolve all conflicts manually, mark them as resolved with
"git add/rm <conflicted_files>", then run "git rebase --continue".
You can instead skip this commit: run "git rebase --skip".
To abort and get back to the state before "git rebase", run "git rebase --abort".
Could not apply 646e0f9cda11... no color ls
```

which looks scary. If you bring up an editor, you'll see it's a typical 3-way merge conflict resolution that you may be familiar with from other source code systems (the rest of ls.c has been omitted):

```
<<<<<<< HEAD
#ifdef COLORLS_NEW
#include <terminfo.h>
=======
#undef COLORLS
#ifdef COLORLS
#include <termcap.h>
>>>>>>> 646e0f9cda11... no color ls
```

The new code is first, and your code is second. The right fix here is to just add a `#undef COLORLS_NEW` before `#ifdef` and then delete the old changes:

```
#undef COLORLS_NEW
#ifdef COLORLS_NEW
#include <terminfo.h>
```

save the file. The rebase was interrupted, so you have to complete it:

```
% git add ls.c
% git rebase --cont
```

which tells git that `ls.c` has changed and to continue the rebase operation. Since there was a conflict, you'll get kicked into the editor to maybe update the commit message.

If you get stuck during the rebase, don't panic. git rebase –abort will take you back to a clean slate. It's important, though, to start with an unmodified tree.

For more on this topic, https://www.freecodecamp.org/news/the-ultimate-guide-to-git-merge-and-git-rebase/ provides a rather extensive treatment. It goes into a lot of cases I didn't cover here for simplicity that are useful to know since they come up from time to time.

5.2 Updating to a New FreeBSD Branch

Let's say you want to main the jump from FreeBSD stable/12 to FreeBSD current. That's easy to do as well, if you have a deep clone.

```
% git checkout main
% # build and install here...
```

and you are done. If you have a local branch, though, there's one or two caveats. First, rebase will rewrite history, so you'll likely want to do something to save it. Second, jumping branches tends to encounter more conflicts. If we pretend the example above was relative to stable/12, then to move to main, I'd suggest the following:

```
% git checkout no-color-ls
% git checkout -b no-color-ls-stable-12 # create another name for this branch
% git rebase -i stable/12 no-color-ls --onto main
```

What the above does is checkout no-color-ls. Then create a new name for it (no-color-ls-stable-12) in case you need to get back to it. Then you rebase onto the main branch. This will find all the commits to the current no-color-ls branch (back to where it meets up with the stable/12 branch) and then it will replay them onto the main branch creating a new no-color-ls branch there (which is why I had you create a placeholder name).

References

[1] Using Git, FreeBSD Handbook, https://docs.freebsd.org/en/books/handbook/mirrors/#git
[2] Git, FreeBSD wiki, https://wiki.freebsd.org/Git
[3] Updating FreeBSD from Source, FreeBSD Handbook, https://docs.freebsd.org/en/books/handbook/cutting-edge/#makeworld.

DREW GURKOWSKI, FreeBSD Foundation