# Package Building Procedures

The FreeBSD Ports Management Team

Revision: 43184

Copyright © 2003, 2004, 2005, 2006, 2007, 2008, 2009, 2010, 2011, 2012, 2013 The FreeBSD Ports Management Team

FreeBSD is a registered trademark of the FreeBSD Foundation.

Intel, Celeron, EtherExpress, i386, i486, Itanium, Pentium, and Xeon are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

SPARC, SPARC64, and UltraSPARC are trademarks of SPARC International, Inc in the United States and other countries. SPARC International, Inc owns all of the SPARC trademarks and under licensing agreements allows the proper use of these trademarks by its members.

Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in this document, and the FreeBSD Project was aware of the trademark claim, the designations have been followed by the "™" or the "®" symbol.

2013-11-13 by hrs.

# Table of Contents

1. Introduction ..... 2
2. Build Client Management ..... 4
3. Jail Build Environment Setup ..... 4
4. Customizing Your Build ..... 5
5. Starting the Build ..... 7
6. Anatomy of a Build ..... 12
7. Build Maintenance ..... 12
8. Monitoring the Build ..... 14
9. Dealing With Build Errors ..... 16
10. Release Builds ..... 16

11. Uploading Packages ..... 17
12. Experimental Patches Builds ..... 19
13. How to configure a new package building node ..... 22
14. How to configure a new FreeBSD branch ..... 32
15. How to delete an unsupported FreeBSD branch ..... 33
16. How to rebase on a supported FreeBSD branch ..... 34
17. How to configure a new architecture ..... 34
18. How to configure a new head node (pointyhat instance) ..... 37
19. Procedures for dealing with disk failures ..... 44

# 1. Introduction

In order to provide pre-compiled binaries of third-party applications for FreeBSD, the Ports Collection is regularly built on one of the "Package Building Clusters." Currently, the main cluster in use is at `http://pointyhat.FreeBSD.org` .

This article documents the internal workings of the cluster.

Note

Many of the details in this article will be of interest only to those on the Ports Management team.

## 1.1. The codebase

Most of the package building magic occurs under the `/a/portbuild` directory. Unless otherwise specified, all paths will be relative to this location. *`${arch}`* will be used to specify one of the package architectures (e.g., amd64, arm, i386™, ia64, powerpc, SPARC64®), and *`${branch}`* will be used to specify the build branch (e.g., 7, 7-exp, 8, 8-exp, 9, 9-exp, 10, 10-exp). The set of branches that `portmgr` currently supports is the same as those that the FreeBSD security team supports.

Note

FreeBSD no longer builds packages for branches 4, 5, or 6, nor for the alpha architecture.

The scripts that control all of this live in either `/a/portbuild/scripts/` or. `/a/portbuild/admin/scripts/` . These are the checked-out copies from the Subversion repository at `base/projects/portbuild/` .

Typically, incremental builds are done that use previous packages as dependencies; this takes less time, and puts less load on the mirrors. Full builds are usually only done:

- right after release time, for the `-STABLE` branches
- periodically to test changes to `-CURRENT`
- for experimental (`"exp-"`) builds

Packages from experimental builds are not uploaded.

## 1.2. Historical notes on the codebase

Until mid-2010, the scripts were completely specific to `pointyhat.FreeBSD.org` as the head (dispatch) node. During the summer of 2010, a significant rewrite was done in order to allow for other hosts to be head nodes. Among the changes were:

- removal of the hard-coding of the string `pointyhat`
- factoring out all configuration constants (which were previously scattered throughout the code) into configuration files (see below)
- appending the hostname to the directories specified by `buildid` (this will allow directories to be unambigious when copied between machines.)
- making the scripts more robust in terms of setting up directories and symlinks
- where necessary, changing certain script invocations to make all the above easier

### Note

Also during this process, the codebase was migrated to the Subversion repository. For reference, the previous version may still be found in CVS.

## 1.3. Notes on privilege separation

As of January 2013, a rewrite is in progress to further separate privileges. The following concepts are introduced:

- Server-side user `portbuild` assumes all responsiblity for operations involving builds and communicating with the clients. This user no longer has access to sudo.
- Server-side user `srcbuild` is created and given responsiblity for operations involving both VCS operations and anything involving src builds for the clients. This user does not have access to sudo.

- The server-side `ports-`*`arch`* users go away.
- None of the above server-side users have ssh keys. Individual `portmgr` will accomplish all those tasks using ksu. (This is still work-in-progress.)
- The only client-side user is also named `portbuild` and still has access to sudo for the purpose of managing jails.

# 2. Build Client Management

You may set up clients to either netboot from the master (*`connected`* nodes) or have them either self-hosted or netboot from some other pxe host (*`disconnected`* nodes). In all cases they set themselves up at boot-time to prepare to build packages.

The cluster master `rsyncs` the interesting data (ports and src trees, bindist tarballs, scripts, etc.) to disconnected nodes during the node-setup phase. Then, the disconnected portbuild directory is nullfs-mounted for jail builds.

The `portbuild` user can ssh(1) to the client nodes to monitor them. Use `sudo` and check the `portbuild.`*`hostname`*`.conf` for the user and access details.

The `scripts/allgohans` script can be used to run a command on all of the *`${arch}`* clients.

# 3. Jail Build Environment Setup

Package builds are performed by the clients in a `jail` populated by the `portbuild` script using the `${arch}/${branch}/builds/${buildid}/bindist.tar` file.

On the server, use the `makeworld` command to build a world from the `${arch}/${branch}/builds/${buildid}/src/` tree and install it into `${arch}/${branch}/builds/${buildid}/bindist.tar`. The tree will be updated first unless `-novcs` is specified.

```
# /a/portbuild/admin/scripts/makeworld ${arch} ${branch} ${buildid} [-novcs]
```

Similiarly on the server, the `bindist.tar` tarball is created from the previously installed world by the `mkbindist` script.

```
# /a/portbuild/admin/scripts/mkbindist ${arch} ${branch} ${buildid}
```

The per-machine tarballs are located on the server in `${arch}/clients`.

The `bindist.tar` file is extracted onto each client at client boot time, and at the start of each pass of the `dopackages` script.

For both commands above, if *${buildid}* is latest, it may be omitted.

Note

Currently the above two scripts must be run as root; otherwise, the install scripts lack sufficient permissions. This is undesirable for security reasons. Work is in progress in -HEAD to allow users to do installations; once that is committed, the intention is to use that and run these two commands as srcbuild.

# 4. Customizing Your Build

You can customize your build by providing local versions of make.conf and/or src.conf, named ${arch}/${branch}/builds/${buildid}/make.conf.server and ${arch}/${branch}/builds/${buildid}/src.conf.server, respectively. These will be used in lieu of the default-named files on the server side.

Similarly, if you wish to also affect the *client-side* make.conf, you may use ${arch}/${branch}/builds/${buildid}/make.conf.client.

Note

Due to the fact that individual clients may each have their own per-host make.conf, the contents of ${arch}/${branch}/builds/${buildid}/make.conf.client will be *appended* to that make.conf, not supplant it, as is done for ${arch}/${branch}/builds/${buildid}/make.conf.server.

Note

There is no similar functionality for ${arch}/${branch}/builds/${buildid}/src.conf.client (what effect would it have?).

### Example 1. Sample **make.conf.target** to test new default ruby version

(For this case, the contents are identical for both server and client.)

```
RUBY_DEFAULT_VER= 1.9
```

### Example 2. Sample **make.conf.target** for clang builds

(For this case, the contents are also identical for both server and client.)

```
.if -!defined(CC) -|| ${CC} == -"cc"
CC=clang
.endif
.if -!defined(CXX) -|| ${CXX} == -"c++"
CXX=clang++
.endif
.if -!defined(CPP) -|| ${CPP} == -"cpp"
CPP=clang-cpp
.endif
# Do not die on warnings
NO_WERROR=
WERROR=
```

### Example 3. Sample **make.conf.server** for pkgng

```
WITH_PKGNG=yes
PKG_BIN=/usr/local/sbin/pkg
```

Example 4. Sample **`make.conf.client`** for pkgng

```
WITH_PKGNG=yes
```

Example 5. Sample **`src.conf.server`** to test new sort codebase

```
WITH_BSD_SORT=yes
```

# 5. Starting the Build

Separate builds for various combinations of architecture and branch are supported. All data private to a build (ports tree, src tree, packages, distfiles, log files, bindist, Makefile, etc) are located under the `${arch}/${branch}/builds/${buildid}/` directory. The most recently created build can be alternatively referenced using buildid `latest`, and the one before using `previous`.

New builds are cloned from the `latest`, which is fast since it uses ZFS.

## 5.1. `dopackages` scripts

The `scripts/dopackages.wrapper` script is used to perform the builds.

```
# dopackages.wrapper ${arch} ${branch} ${buildid} [-options]
```

Most often, you will be using `latest` for the value of *`buildid`*.

`[-options]` may be zero or more of the following:

- `-keep` - Do not delete this build in the future, when it would be normally deleted as part of the `latest` - `previous` cycle. Do not forget to clean it up manually when you no longer need it.
- `-nofinish` - Do not perform post-processing once the build is complete. Useful if you expect that the build will need to be restarted once it finishes. If you use this option, do not forget to cleanup the clients when you do not need the build any more.
- `-finish` - Perform post-processing only.

- `-nocleanup` - By default, when the `-finish` stage of the build is complete, the build data will be deleted from the clients. This option will prevent that.
- `-restart` - Restart an interrupted (or non-`finish`ed) build from the beginning. Ports that failed on the previous build will be rebuilt.
- `-continue` - Restart an interrupted (or non-`finish`ed) build. Will not rebuild ports that failed on the previous build.
- `-incremental` - Compare the interesting fields of the new `INDEX` with the previous one, remove packages and log files for the old ports that have changed, and rebuild the rest. This cuts down on build times substantially since unchanged ports do not get rebuilt every time.
- `-cdrom` - This package build is intended to end up on a CD-ROM, so `NO_CDROM` packages and distfiles should be deleted in post-processing.
- `-nobuild` - Perform all the preprocessing steps, but do not actually do the package build.
- `-noindex` - Do not rebuild `INDEX` during preprocessing.
- `-noduds` - Do not rebuild the `duds` file (ports that are never built, e.g., those marked `IGNORE`, `NO_PACKAGE` , etc.) during preprocessing.
- `-nochecksubdirs` - Do not check the `SUBDIRS` for ports that are not connected to the build.
- `-trybroken` - Try to build `BROKEN` ports (off by default because the amd64/i386™ clusters are fast enough now that when doing incremental builds, more time was spent rebuilding things that were going to fail anyway. Conversely, the other clusters are slow enough that it would be a waste of time to try and build `BROKEN` ports).

### Note

With `-trybroken`, you probably also want to use `-fetch-original` and `-unlimited-errors`.

- `-nosrc` - Do not update the `src` tree from the ZFS snapshot, keep the tree from previous build instead.
- `-srcvcs` - Do not update the `src` tree from the ZFS snapshot, update it with a fresh checkout instead.

- `-noports` - Do not update the `ports` tree from the ZFS snapshot, keep the tree from previous build instead.
- `-portsvcs` - Do not update the `ports` tree from the ZFS snapshot, update it with a fresh checkout instead.
- `-norestr` - Do not attempt to build `RESTRICTED` ports.
- `-noplistcheck` - Do not make it fatal for ports to leave behind files after deinstallation.
- `-nodistfiles` - Do not collect distfiles that pass `make checksum` for later uploading to `ftp-master`.
- `-fetch-original` - Fetch the distfile from the original `MASTER_SITES` rather than any cache such as on `ftp-master`.
- `-unlimited-errors` - defeat the "qmanager threshhold" check for runaway builds. You want this primarily when doing a `-restart` of a build that you expect to mostly fail, or perhaps a `-trybroken` run. By default, the threshhold check is done.

Unless you specify `-restart`, `-continue`, or `-finish`, the symlinks for the existing builds will be rotated. i.e, the existing symlink for `previous` will be deleted; the most recent build will have its symlink changed to `previous/`; and a new build will be created and symlinked into `latest/`.

If the last build finished cleanly you do not need to delete anything. If it was interrupted, or you selected `-nocleanup`, you need to clean up clients by running

```
% build cleanup ${arch} ${branch} ${buildid} --full
```

When a new build is created, the directories `errors/`, `logs/`, `packages/`, and so forth, are cleaned by the scripts. If you are short of space, you can also clean out `ports/distfiles/`. Leave the `latest/` directory alone; it is a symlink for the webserver.

### Note

`dosetupnodes` is supposed to be run from the `dopackages` script in the `-restart` case, but it can be a good idea to run it by hand and then verify that the clients all have the expected job load. Sometimes, `dosetupnode` cannot clean up a build and you need to do it by hand. (This is a bug.)

Make sure the *`${arch}`* build is run as the `portbuild` user or it will complain loudly.

### Note

The actual package build itself occurs in two identical phases. The reason for this is that sometimes transient problems (e.g., NFS failures, FTP sites being unreachable, etc.) may halt a build. Doing things in two phases is a workaround for these types of problems.

Be careful that `ports/Makefile` does not specify any empty subdirectories. This is especially important if you are doing an -exp build. If the build process encounters an empty subdirectory, both package build phases will stop short, and an error similar to the following will be written to `${arch}/${branch}/journal` :

```
don't know how to make dns-all(continuing)
```

To correct this problem, simply comment out or remove the `SUBDIR` entries that point to empty subdirectories. After doing this, you can restart the build by running the proper `dopackages` command with the `-restart` option.

### Note

This problem also appears if you create a new category `Makefile` with no `SUBDIR`s in it. This is probably a bug.

### Example 6. Update the i386-7 tree and do a complete build

```
% dopackages.wrapper i386 8 latest --nosrc --norestr --nofinish
```

Example 7. Restart an interrupted amd64-8 build without updating

```
% dopackages.wrapper amd64 8 latest --nosrc --noports --
norestr --continue --noindex --noduds --nofinish
```

Example 8. Post-process a completed sparc64-8 tree

```
% dopackages.wrapper sparc64 8 --finish
```

Hint: it is usually best to run the `dopackages` command inside of `screen(1)`.

## 5.2. `build` command

You may need to manipulate the build data before starting it, especially for experimental builds. This is done with the `build` command. Here are the useful options for creation:

- `build create arch branch [newid]` - Creates *`newid`* (or a datestamp if not specified). Only needed when bringing up a new branch or a new architecture.
- `build clone arch branch oldid [newid]` - Clones *`oldid`* to *`newid`* (or a datestamp if not specified).
- `build srcupdate arch branch buildid` - Replaces the src tree with a new ZFS snapshot. Do not forget to use `-nosrc` flag to `dopackages` later!
- `build portsupdate arch branch buildid` - Replaces the ports tree with a new ZFS snapshot. Do not forget to use `-noports` flag to `dopackages` later!

## 5.3. Building a single package

Sometimes there is a need to rebuild a single package from the package set. This can be accomplished with the following invocation:

```
# path/qmanager/packagebuild amd64 7-exp 20080904212103 ↺
aclock-0.2.3_2.tbz
```

# 6. Anatomy of a Build

A full build without any `-no` options performs the following operations in the specified order:

1. An update of the current `ports` tree from the ZFS snapshot[1]
2. An update of the running branch's `src` tree from the ZFS snapshot[1]
3. Checks which ports do not have a `SUBDIR` entry in their respective category's `Makefile`[1]
4. Creates the `duds` file, which is a list of ports not to build[12]
5. Generates a fresh `INDEX` file[12]
6. Sets up the nodes that will be used in the build[12]
7. Builds a list of restricted ports[12]
8. Builds packages (phase 1)[3]
9. Performs another node setup[1]
10. Builds packages (phase 2)[3]

# 7. Build Maintenance

There are several cases where you will need to manually clean up a build:

1. You have manually interrupted it.
2. The head node has been rebooted while a build was running.
3. `qmanager` has crashed and has been restarted.

## 7.1. Interrupting a Build

Manually interrupting a build is a bit messy. First you need to identify the tty in which it's running (either record the output of tty(1) when you start the build, or use `ps x` to identify it. You need to make sure that nothing else important is running in this tty, e.g., **`ps -t p1`** or whatever. If there is not, you can just kill off the whole term easily with

[1] Status of these steps can be found in `${arch}/${branch}/build.log` as well as on stderr of the tty running the `dopackages` command.

[2] If any of these steps fail, the build will stop cold in its tracks.

[3] Status of these steps can be found in `${arch}/${branch}/journal`. Individual ports will write their build logs to `${arch}/${branch}/logs/` and their error logs to `${arch}/${branch}/errors/`.

**pkill -t pts/1** ; otherwise issue a **kill -HUP** in there by, for example, **ps -t pts/1 -o pid= | xargs kill -HUP** . Replace *p1* by whatever the tty is, of course.

The package builds dispatched by `make` to the client machines will clean themselves up after a few minutes (check with `ps x` until they all go away).

If you do not kill make(1), then it will spawn more jobs. If you do not kill `dopackages`, then it will restart the entire build. If you do not kill the `pdispatch` processes, they'll keep going (or respawn) until they've built their package.

## 7.2. Cleaning up a Build

To free up resources, you will need to clean up client machines by running `build cleanup` command. For example:

```
% /a/portbuild/scripts/build cleanup i386 8-exp 20080714120411 --full
```

If you forget to do this, then the old build `jails` will not be cleaned up for 24 hours, and no new jobs will be dispatched in their place since `pointyhat` thinks the job slot is still occupied.

To check, **cat ~/loads/*** to display the status of client machines; the first column is the number of jobs it thinks is running, and this should be roughly concordant with the load average. `loads` is refreshed every 2 minutes. If you do **ps x | grep pdispatch** and it is less than the number of jobs that `loads` thinks are in use, you are in trouble.

### Note

The following notes about mounting only apply to `connected` nodes.

You may have problem with the `umount` commands hanging. If so, you are going to have to use the `allgohans` script to run an ssh(1) command across all clients for that buildenv. For example:

```
% ssh gohan24 df
```

will get you a df, and

```
% allgohans -"umount --f pointyhat.freebsd.org:/var/portbuild/i386/8-
exp/ports"
% allgohans -"umount --f pointyhat.freebsd.org:/var/portbuild/i386/8-
exp/src"
```

are supposed to get rid of the hanging mounts. You will have to keep doing them since there can be multiple mounts.

### Note

Ignore the following:

```
umount: pointyhat.freebsd.org:/var/portbuild/i386/8-
exp/ports: statfs: No such file or directory
umount: pointyhat.freebsd.org:/var/portbuild/i386/8-
exp/ports: unknown file system
umount: Cleanup of -/x/tmp/8-exp/chroot/53837/
compat/linux/proc failed!
/x/tmp/8-exp/chroot/53837/compat/linux/proc: not a ↺
file system root directory
```

The former two mean that the client did not have those mounted; the latter two are a bug.

You may also see messages about `procfs`.

### Note

The above is the end of the notes that apply only to `connected` nodes.

After you have done all the above, remove the `${arch}/lock` file before trying to restart the build. If you do not, `dopackages` will simply exit.

If you have to do a ports tree update before restarting, you may have to rebuild either `duds`, `INDEX`, or both.

## 7.3. Maintaining builds with the `build` command

Here are the rest of the options for the `build` command:

- `build destroy arch branch` - Destroy the build id.
- `build list arch branch` - Shows the current set of build ids.

# 8. Monitoring the Build

You can use `qclient` command to monitor the status of build nodes, and to list the currently scheduled jobs:

```
% python path/qmanager/qclient jobs
% python path/qmanager/qclient status
```

The **scripts/stats ${branch}** command shows the number of packages already built.

Running **cat /a/portbuild/*/loads/*** shows the client loads and number of concurrent builds in progress. The files that have been recently updated are the clients that are online; the others are the offline clients.

### Note

The `pdispatch` command does the dispatching of work onto the client, and post-processing. `ptimeout.host` is a watchdog that kills a build after timeouts. So, having 50 `pdispatch` processes but only 4 ssh(1) processes means 46 `pdispatch`es are idle, waiting to get an idle node.

Running **tail -f ${arch}/${branch}/build.log** shows the overall build progress.

If a port build is failing, and it is not immediately obvious from the log as to why, you can preserve the `WRKDIR` for further analysis. To do this, touch a file called `.keep` in the port's directory. The next time the cluster tries to build this port, it will tar, compress, and copy the `WRKDIR` to `${arch}/${branch}/wrkdirs/` .

If you find that the system is looping trying to build the same package over and over again, you may be able to fix the problem by rebuilding the offending package by hand.

If all the builds start failing with complaints that they cannot load the dependent packages, check to see that httpd is still running, and restart it if not.

Keep an eye on df(1) output. If the `/a/portbuild` file system becomes full then Bad Things™ happen.

The status of all current builds is generated periodically into the `packagestats.html` file, e.g., `http://pointyhat.FreeBSD.org/errorlogs/packagestats.html` . For each `buildenv` , the following is displayed:

- `updated` is the contents of `.updated`. This is why we recommend that you update `.updated` for `-exp` runs (see below).
- date of `latest log`
- number of lines in `INDEX`
- the number of current `build logs`

- the number of completed `packages`
- the number of `errors`
- the number of duds (shown as `skipped`)
- `missing` shows the difference between `INDEX` and the other columns. If you have restarted a run after a ports tree update, there will likely be duplicates in the packages and error columns, and this column will be meaningless. (The script is naive).
- `running` and `completed` are guesses based on a grep(1) of `build.log`.

# 9. Dealing With Build Errors

The easiest way to track build failures is to receive the emailed logs and sort them to a folder, so you can maintain a running list of current failures and detect new ones easily. To do this, add an email address to `${branch}/portbuild.conf`. You can easily bounce the new ones to maintainers.

After a port appears broken on every build combination multiple times, it is time to mark it `BROKEN`. Two weeks' notification for the maintainers seems fair.

Note

To avoid build errors with ports that need to be manually fetched, put the distfiles into `~ftp/pub/FreeBSD/distfiles`. Access restrictions are in place to make sure that only the build clients can access this directory.

# 10. Release Builds

When building packages for a release, it may be necessary to manually update the `ports` and `src` trees to the release tag and use `-novcs` and `-noportsvcs`.

To build package sets intended for use on a CD-ROM, use the `-cdrom` option to `dopackages`.

If the disk space is not available on the cluster, use `-nodistfiles` to avoid collecting distfiles.

After the initial build completes, restart the build with `-restart -fetch-original` to collect updated distfiles as well. Then, once the build is post-processed, take an inventory of the list of files fetched:

```
% cd ${arch}/${branch}
% find distfiles > distfiles-${release}
```

You should use that output to periodically clean out the distfiles from `ftp-master`. When space gets tight, distfiles from recent releases can be kept while others can be thrown away.

Once the distfiles have been uploaded (see below), the final release package set must be created. Just to be on the safe side, run the `${arch}/${branch}/cdrom.sh` script by hand to make sure all the CD-ROM restricted packages and distfiles have been pruned. Then, copy the `${arch}/${branch}/packages` directory to `${arch}/${branch}/packages-${release}`. Once the packages are safely moved off, contact the Release Engineering Team <re@FreeBSD.org> and inform them of the release package location.

Remember to coordinate with the Release Engineering Team <re@FreeBSD.org> about the timing and status of the release builds.

# 11. Uploading Packages

### Note

For FreeBSD.org as of 2013, the instructions about uploading to `ftp-master` are obsolete. In the future, `ftp-master` will pull from `pointyhat`, using a mechanism yet to be implemented. However, the instructions about `RESTRICTED` and `NO_CDROM` must still be *carefully* followed.

Once a build has completed, packages and/or distfiles can be transferred to `ftp-master` for propagation to the FTP mirror network. If the build was run with `-nofinish`, then make sure to follow up with `dopackages -finish` to post-process the packages (removes `RESTRICTED` and `NO_CDROM` packages where appropriate, prunes packages not listed in `INDEX`, removes from `INDEX` references to packages not built, and generates a `CHECKSUM.MD5` summary); and distfiles (moves them from the temporary `distfiles/.pbtmp` directory into `distfiles/` and removes `RESTRICTED` and `NO_CDROM` distfiles).

It is usually a good idea to run the `restricted.sh` and/or `cdrom.sh` scripts by hand after `dopackages` finishes just to be safe. Run the `restricted.sh` script before uploading to `ftp-master`, then run `cdrom.sh` before preparing the final package set for a release.

The package subdirectories are named by whether they are for `release`, `stable`, or `current`. Examples:

- `packages-7.2-release`

- packages-7-stable
- packages-8-stable
- packages-9-stable
- packages-10-current

### Note

Some of the directories on ftp-master are, in fact, symlinks. Examples:

- packages-stable
- packages-current

Be sure you move the new packages directory over the *real* destination directory, and not one of the symlinks that points to it.

If you are doing a completely new package set (e.g., for a new release), copy packages to the staging area on ftp-master with something like the following:

```
# cd -/a/portbuild/${arch}/${branch}
# tar cfv -- packages/ -| ssh portmgr@ftp-master tar xfC -- w/ports/
${arch}/tmp/${subdir}
```

Then log into ftp-master, verify that the package set was transferred successfully, remove the package set that the new package set is to replace (in ~/w/ports/${arch} ), and move the new set into place. (w/ is merely a shortcut.)

For incremental builds, packages should be uploaded using rsync so we do not put too much strain on the mirrors.

*ALWAYS* use -n first with rsync and check the output to make sure it is sane. If it looks good, re-run the rsync without the -n option.

Example rsync command for incremental package upload:

```
# rsync --n --r --v --l --t --p ---delete packages/ portmgr@ftp-
master:w/ports/${arch}/${subdir}/ -| tee log
```

Distfiles should be transferred with the cpdistfiles script:

```
# /a/portbuild/scripts/cpdistfiles ${arch} ${branch} ${buildid} [-
yesreally] -| tee log2
```

Doing it by hand is deprecated.

# 12. Experimental Patches Builds

Note

Most of the information in this section is obsolete as of 2013 and needs to be rewritten.

Experimental patches builds are run from time to time to new features or bugfixes to the ports infrastructure (i.e. `bsd.port.mk` ), or to test large sweeping upgrades. At any given time there may be several simultaneous experimental patches branches, such as `8-exp` on the amd64 architecture.

In general, an experimental patches build is run the same way as any other build, except that you should first update the ports tree to the latest version and then apply your patches. To do the former, you can use the following:

Note

The following example is obsolete

```
% cvs --R update --dP > update.out
% date > -.updated
```

This will most closely simulate what the `dopackages` script does. (While `.updated` is merely informative, it can be a help.)

You will need to edit `update.out` to look for lines beginning with `^M`, `^C`, or `^?` and then deal with them.

It is always a good idea to save original copies of all changed files, as well as a list of what you are changing. You can then look back on this list when doing the final commit, to make sure you are committing exactly what you tested.

Since the machine is shared, someone else may delete your changes by mistake, so keep a copy of them in e.g., your home directory on `freefall`. Do not use `tmp/`; since `pointyhat` itself runs some version of `-CURRENT`, you can expect reboots (if nothing else, for updates).

In order to have a good control case with which to compare failures, you should first do a package build of the branch on which the experimental patches branch is based for the i386™ architecture (currently this is `8`). Then, when preparing for the experimental

patches build, checkout a ports tree and a src tree with the same date as was used for the control build. This will ensure an apples-to-apples comparison later.

Once the build finishes, compare the control build failures to those of the experimental patches build. Use the following commands to facilitate this (this assumes the `8` branch is the control branch, and the `8-exp` branch is the experimental patches branch):

```
% cd -/a/portbuild/i386/8-exp/errors
% find -. --name \*.log\* -| sort > -/tmp/8-exp-errs
% cd -/a/portbuild/i386/8/errors
% find -. --name \*.log\* -| sort > -/tmp/8-errs
```

### Note

If it has been a long time since one of the builds finished, the logs may have been automatically compressed with bzip2. In that case, you must use **`sort | sed 's,\.bz2,,g'`** instead.

```
% comm --3 -/tmp/8-errs -/tmp/8-exp-errs -| less
```

This last command will produce a two-column report. The first column is ports that failed on the control build but not in the experimental patches build; the second column is vice versa. Reasons that the port might be in the first column include:

- Port was fixed since the control build was run, or was upgraded to a newer version that is also broken (thus the newer version should appear in the second column)
- Port is fixed by the patches in the experimental patches build
- Port did not build under the experimental patches build due to a dependency failure

Reasons for a port appearing in the second column include:

- Port was broken by the experimental patches
- Port was upgraded since the control build and has become broken
- Port was broken due to a transient error (e.g., FTP site down, package client error, etc.)

Both columns should be investigated and the reason for the errors understood before committing the experimental patches set. To differentiate between broken by experimental patches and broken by upgrading above, you can do a rebuild of the affected packages under the control branch:

```
% cd -/a/portbuild/i386/8/ports
```

### Note

The following example is obsolete

### Note

Be sure to **cvs update** this tree to the same date as the experimental patches tree.

The following command will set up the control branch for the partial build:

```
% /a/portbuild/scripts/dopackages.wrapper i386 8 latest --
noportsvcs --nobuild --novcs --nofinish
```

The builds must be performed from the `packages/All` directory. This directory should initially be empty except for the Makefile symlink. If this symlink does not exist, it must be created:

```
% cd -/a/portbuild/i386/8/packages/All
% ln --sf -../../Makefile -.
% make --k --j<#> <list of packages to build>
```

### Note

<#> is the concurrency of the build to attempt. It is usually the sum of the weights listed in `/a/portbuild/i386/mlist` unless you have a reason to run a heavier or lighter build.

The list of packages to build should be a list of package names (including versions) as they appear in `INDEX`. The `PKGSUFFIX` (i.e., `.tgz` or `.tbz`) is optional.

This will build only those packages listed as well as all of their dependencies.

You can check the progress of this partial build the same way you would a regular build.

Once all the errors have been resolved, you can commit the package set. After committing, it is customary to send a `HEADS UP` email to ports@FreeBSD.org and copy ports-

developers@FreeBSD.org informing people of the changes. A summary of all changes should also be committed to `/usr/ports/CHANGES` .

# 13. How to configure a new package building node

Before following these steps, please coordinate with `portmgr`.

## 13.1. Node requirements

### Note

This section is only of interest when considering tier-2 architectures.

Here are the requirement for what a node needs to be generally useful.

- CPU capacity: anything less than 500MHz is generally not useful for package building.

### Note

We are able to adjust the number of jobs dispatched to each machine, and we generally tune the number to use 100% of CPU.

- RAM: Less than 2G is not very useful; 8G or more is preferred. We have been tuning to one job per 512M of RAM.
- disk: at least 20G is needed for filesystem; 32G is needed for swap. Best performance will be if multiple disks are used, and configured as `geom` stripes. Performance numbers are also TBA.

### Note

Package building will test disk drives to destruction. Be aware of what you are signing up for!

- network bandwidth: TBA. However, an 8-job machine has been shown to saturate a cable modem line.

## 13.2. Preparation

1. Pick a unique hostname. It does not have to be a publicly resolvable hostname (it can be a name on your internal network).

2. By default, package building requires the following TCP ports to be accessible: 22 (`ssh`), 414 (`infoseek`), and 8649 (`ganglia`). If these are not accessible, pick others and ensure that an `ssh` tunnel is set up (see below).

   (Note: if you have more than one machine at your site, you will need an individual TCP port for each service on each machine, and thus `ssh` tunnels will be necessary. As such, you will probably need to configure port forwarding on your firewall.)

3. Decide if you will be booting natively or via `pxeboot`. You will find that it is easier to keep up with changes to `-current` with the latter, especially if you have multiple machines at your site.

4. Pick a directory to hold ports configuration and `chroot` subdirectories. It may be best to put it this on its own partition. (Example: `/usr2/` .)

   **Note**

   The filename `chroot` is a historical remnant. The `chroot` command is no longer used.

5. Decide if you will be using a local squid cache on the client, instead of the server. It is more efficient to run it on the server. If you are doing that, skip the "squid" steps below.)

## 13.3. Configuring `src`

1. Create a directory to contain the latest `-current` source tree and check it out. (Since your machine will likely be asked to build packages for `-current`, the kernel it runs should be reasonably up-to-date with the `bindist` that will be exported by our scripts.)

2. If you are using `pxeboot`: create a directory to contain the install bits. You will probably want to use a subdirectory of `/pxeroot`, e.g., `/pxeroot/${arch}-${branch}` . Export that as `DESTDIR`.

3. If you are cross-building, export `TARGET_ARCH` =*`${arch}`* .

Note

The procedure for cross-building ports is not yet defined.

4. Generate a kernel config file. Include GENERIC (or, if on i386™, and you are using more than 3.5G, PAE).

   Required options:

```
options         NULLFS
options         TMPFS
```

   Suggested options:

```
options         GEOM_CONCAT
options         GEOM_STRIPE
options         SHMMAXPGS=65536
options         SEMMNI=40
options         SEMMNS=240
options         SEMUME=40
options         SEMMNU=120
```

   If you are interested in debugging general problems, you may wish to use the following. However, for unattended operations, it is best to leave it out:

```
options         ALT_BREAK_TO_DEBUGGER
```

   For PAE, it is not currently possible to load modules. Therefore, if you are running an architecture that supports Linux emulation, you will need to add:

```
options         COMPAT_LINUX
options         LINPROCFS
```

   Also for PAE, as of 20110912 you need the following. This needs to be investigated:

```
nooption        NFSD                    # New Network Filesystem ↺
Server
options         NFSCLIENT               # Network Filesystem ↺
Client
options         NFSSERVER               # Network Filesystem ↺
Server
```

5. As root, do the usual build steps, e.g.:

```
# make --j4 buildworld
```

```
# make buildkernel KERNCONF=${kernconf}
# make installkernel KERNCONF=${kernconf}
# make installworld
```

The install steps use `DESTDIR`.

6. Customize files in `etc/`. Whether you do this on the client itself, or another machine, will depend on whether you are using `pxeboot`.

   If you are using `pxeboot`: create a subdirectory of `${DESTDIR}` called `conf/`. Create one subdirectory `default/etc/`, and (if your site will host multiple nodes), subdirectories `${ip-address}/etc/` to contain override files for individual hosts. (You may find it handy to symlink each of those directories to a hostname.) Copy the entire contents of `${DESTDIR}/etc/` to `default/etc/`; that is where you will edit your files. The by-ip-address `etc/` directories will probably only need customized `rc.conf` files.

   In either case, apply the following steps:

   - Create a `portbuild` user and group. It can have the `'*'` password.

     Create `/home/portbuild/.ssh/` and populate `authorized_keys`.

   - If you are using ganglia for monitoring, add the following user:

     ```
     ganglia:*:102:102::0:0:User &:/usr/local/ganglia:/bin/sh
     ```

     Add it to `etc/group` as well.

   - If you are using a local squid cache on the client, add the following user:

     ```
     squid:*:100:100::0:0:User &:/usr/local/squid:/bin/sh
     ```

     Add it to `etc/group` as well.

   - Create the appropriate files in `etc/.ssh/`.

   - In `etc/crontab`: add

     ```
     *       *       *       *       *       root    -/var/portbuild/
     scripts/client-metrics
     ```

   - Create the appropriate `etc/fstab`. (If you have multiple, different, machines, you will need to put those in the override directories.)

   - In `etc/inetd.conf`: add

     ```
     infoseek        stream  tcp     nowait  nobody  -/var/portbuild/
     scripts/reportload
     ```

- You should run the cluster on UTC. If you have not set the clock to UTC:

```
# cp --p -/usr/share/zoneinfo/Etc/UTC etc/localtime
```

- Create the appropriate etc/rc.conf . (If you are using pxeboot, and have multiple, different, machines, you will need to put those in the override directories.)

  Recommended entries for physical nodes:

```
hostname="${hostname} "
inetd_enable="YES"
linux_enable="YES"
nfs_client_enable="YES"
ntpd_enable="YES"
sendmail_enable="NONE"
sshd_enable="YES"
sshd_program="/usr/local/sbin/sshd"
```

  If you are using ganglia for monitoring, add the following

```
gmond_enable="YES"
```

  If you are using a local squid cache on the client, add the following

```
squid_enable="YES"
squid_chdir="/a/squid/logs "
squid_pidfile="/a/squid/logs/squid.pid  "
```

  Required entries for VMWare-based nodes:

```
vmware_guest_vmmemctl_enable="YES"
vmware_guest_guestd_enable="YES"
```

  Recommended entries for VMWare-based nodes:

```
hostname=""
ifconfig_em0="DHCP"
fsck_y_enable="YES"

inetd_enable="YES"
linux_enable="YES"
nfs_client_enable="YES"
sendmail_enable="NONE"
sshd_enable="YES"
sshd_program="/usr/local/sbin/sshd"

gmond_enable="YES"
squid_enable="YES"
squid_chdir="/a/squid/logs "
squid_pidfile="/a/squid/logs/squid.pid  "
```

ntpd(8) should *not* be enabled for VMWare instances.

Also, it may be possible to leave squid disabled by default so as to not have `/a` persistent (which should save instantiation time.) Work is still ongoing.

- Create `etc/resolv.conf` , if necessary.
- Modify `etc/sysctl.conf` :

```
9a10,30
> kern.corefile=/a/%N.core
> kern.sugid_coredump=1
> #debug.witness_ddb=0
> #debug.witness_watch=0
>
> # squid needs a lot of fds (leak?)
> kern.maxfiles=40000
> kern.maxfilesperproc=30000
>
> # Since the NFS root is static we do not need to check ↺
frequently for file changes
> # This saves >75% of NFS traffic
> vfs.nfs.access_cache_timeout=300
> debug.debugger_on_panic=1
>
> # For jailing
> security.jail.sysvipc_allowed=1
> security.jail.allow_raw_sockets=1
> security.jail.chflags_allowed=1
> security.jail.enforce_statfs=1
>
> vfs.lookup_shared=1
```

- If desired, modify `etc/syslog.conf` to change the logging destinations to `@pointyhat.freebsd.org` .

## 13.4. Configuring `ports`

1. Install the following ports:

```
net/rsync
security/sudo
```

You may also wish to install:

```
security/openssh-portable (with HPN on)
```

If you are using ganglia for monitoring, install the following:

```
sysutils/ganglia-monitor-core (with GMETAD off)
```

If you are using a local squid cache on the client, install the following

```
www/squid31 (with SQUID_AUFS on)
```

2. Customize files in `usr/local/etc/` . Whether you do this on the client itself, or another machine, will depend on whether you are using `pxeboot`.

### Note

The trick of using `conf` override subdirectories is less effective here, because you would need to copy over all subdirectories of `usr/`. This is an implementation detail of how the pxeboot works.

Apply the following steps:

- If you are using ganglia, modify `usr/local/etc/gmond.conf` :

```
21,22c21,22
<   name = -"unspecified"
<   owner = -"unspecified"
---
>   name = -"${arch} package build cluster"
>   owner = -"portmgr@FreeBSD.org"
24c24
<   url = -"unspecified"
---
>   url = -"http://pointyhat.freebsd.org"
```

If there are machines from more than one cluster in the same multicast domain (basically = LAN) then change the multicast groups to different values (.71, .72, etc).

- Create `usr/local/etc/rc.d/portbuild.sh` , using the appropriate value for `scratchdir` :

```
#!/bin/sh
#
# Configure a package build system post-boot

scratchdir=/a

ln --sf ${scratchdir}/portbuild -/var/

# Identify builds ready for use
cd -/var/portbuild/arch
for i in */builds/*; do
```

```
    if [ --f ${i}/.ready -]; then
        mkdir -/tmp/.setup-${i##*/}
    fi
done

# Flag that we are ready to accept jobs
touch -/tmp/.boot_finished
```

- If you are using a local squid cache, modify, `usr/local/etc/squid/squid.conf` :

```
288,290c288,290
< #auth_param basic children 5
< #auth_param basic realm Squid proxy-caching web server
< #auth_param basic credentialsttl 2 hours
---
> auth_param basic children 5
> auth_param basic realm Squid proxy-caching web server
> auth_param basic credentialsttl 2 hours
611a612
> acl localnet src 127.0.0.0/255.0.0.0
655a657
> http_access allow localnet
2007a2011
> maximum_object_size 400 MB
2828a2838
> negative_ttl 0 minutes
```

  Also, change `usr/local` to `usr2` in `cache_dir` , `access_log` , `cache_log` , `cache_store_log` , `pid_filename` , `netdb_filename` , `coredump_dir` .

  Finally, change the `cache_dir` storage scheme from `ufs` to `aufs` (offers better performance).

- Configure `ssh`: copy `etc/ssh` to `usr/local/etc/ssh` and add `NoneEnabled yes` to `sshd_config` .

-

  > **Note**
  >
  > This step is under review.

  Create `usr/local/etc/sudoers/sudoers.d/portbuild` :

```
# local changes for package building
portbuild     ALL=(ALL) NOPASSWD: ALL
```

## 13.5. Configuration on the client itself

1. Change into the port/package directory you picked above, e.g., `cd /usr2`.

2. As root:

```
# mkdir portbuild
# chown portbuild:portbuild portbuild
# mkdir pkgbuild
# chown portbuild:portbuild pkgbuild
```

   If you are using a local squid cache:

```
# mkdir squid
# mkdir squid/cache
# mkdir squid/logs
# chown --R squid:squid squid
```

3. If clients preserve `/var/portbuild` between boots then they must either preserve their `/tmp`, or revalidate their available builds at boot time (see the script on the `amd64` machines). They must also clean up stale jails from previous builds before creating `/tmp/.boot_finished` .

4. Boot the client.

5. If you are using a local squid cache, as root, initialize the `squid` directories:

```
squid --z
```

## 13.6. Configuration on the server

These steps need to be taken by a `portmgr` acting as `portbuild` on the server.

1. If any of the default TCP ports is not available (see above), you will need to create an `ssh` tunnel for them and include its invocation command in `portbuild` 's `crontab`.

2. Unless you can use the defaults, add an entry to `/home/portbuild/.ssh/config` to specify the public IP address, TCP port for `ssh`, username, and any other necessary information.

3. Create `/a/portbuild/${arch}/clients/bindist-${hostname}.tar` .

   - Copy one of the existing ones as a template and unpack it in a temporary directory.
   - Customize `etc/resolv.conf` for the local site.
   - Customize `etc/make.conf` for FTP fetches for the local site. Note: the nulling-out of `MASTER_SITE_BACKUP` must be common to all nodes, but the first entry in `MASTER_SITE_OVERRIDE` should be the nearest local FTP mirror. Example:

```
.if defined(FETCH_ORIGINAL)
```

```
MASTER_SITE_BACKUP=
.else
MASTER_SITE_OVERRIDE= \
 ftp://friendly-local-ftp-mirror /pub/FreeBSD/ports/distfiles/
${DIST_SUBDIR}/ \
 ftp://${BACKUP_FTP_SITE}/pub/FreeBSD/distfiles/${DIST_SUBDIR}/
.endif
```

- `tar` it up and move it to the right location.

Hint: you will need one of these for each machine; however, if you have multiple machines at one site, you should create a site-specific one (e.g., in `/a/portbuild/conf/clients/` ) and symlink to it.

4. Create `/a/portbuild/${arch}/portbuild-${hostname}` using one of the existing ones as a guide. This file contains overrides to `/a/portbuild/${arch}/portbuild.conf` .

   Suggested values:

```
disconnected=1
scratchdir=/usr2/pkgbuild
client_user=portbuild
sudo_cmd="sudo --H"
rsync_gzip=-z

infoseek_host=localhost
infoseek_port=${tunelled-tcp-port}
```

   If you will be using squid on the client:

```
http_proxy="http://localhost:3128/"
squid_dir=/usr2/squid
```

   If, instead, you will be using squid on the server:

```

http_proxy="http://servername :3128/"
```

   Possible other values:

```
use_md_swap=1
md_size=9g
use_zfs=1
scp_cmd="/usr/local/bin/scp"
ssh_cmd="/usr/local/bin/ssh"
```

These steps need to be taken by a `portmgr` acting as `root` on `pointyhat`.

1. Add the public IP address to `/etc/hosts.allow` . (Remember, multiple machines can be on the same IP address.)

2. If you are using ganglia, add an appropriate `data_source` entry to `/usr/local/etc/gmetad.conf`:

   ```
   data_source -"arch/location Package Build Cluster" 30 hostname
   ```

   You will need to restart `gmetad`.

## 13.7. Enabling the node

These steps need to be taken by a `portmgr` acting as `portbuild`:

1. Ensure that `ssh` to the client is working by executing **`ssh hostname uname -a`**. The actual command is not important; what is important is to confirm the setup, and also add an entry into `known_hosts`, once you have confirmed the node's identity.

2. Populate the client's copy of `/var/portbuild/scripts/` by something like **`/a/portbuild/scripts/dosetupnode arch major latest hostname`**. Verify that you now have files in that directory.

3. Test the other TCP ports by executing **`telnet hostname portnumber`**. 414 (or its tunnel) should give you a few lines of status information including `arch` and `osversion`; `8649` should give you an `XML` response from `ganglia`.

This step needs to be taken by a `portmgr` acting as `portbuild`:

- Tell `qmanager` about the node. Example:

  ```
  python path/qmanager/qclient add name=uniquename arch=arch
  osversion=osversion numcpus=number haszfs=0 online=1 domain=domain
  primarypool=package pools="package all" maxjobs=1 acl="ports-
  arch,deny_all"
  ```

Finally, again as `portmgr` acting as `portbuild`:

- Once you are sure that the client is working, tell pollmachine about it by adding it to `/a/portbuild/${arch}/mlist`.

# 14. How to configure a new FreeBSD branch

## 14.1. Steps necessary before qmanager is started

When a new branch is created, some work needs to be done to specify that the previous branch is no longer equivalent to `HEAD`.

Note

As srcbuild:

- Edit /a/portbuild/conf/admin/admin.conf with the following changes:
  - Add *new-branch* to SRC_BRANCHES .
  - For what was previously head, change SRC_BRANCH_ *branch* _SUBDIR to releng/branch.0 (literal zero).
  - Add SRC_BRANCH_ *new-branch* _SUBDIR =head.
- Run /a/portbuild/admin/scripts/updatesnap manually.

### 14.2. Steps necessary after qmanager is started

- For each branch that will be supported, do the following:
  - As portbuild, kick-start the build for the branch with:

    ```
    build create arch branch
    ```

  - As srcbuild, create bindist.tar .

## 15. How to delete an unsupported FreeBSD branch

When an old branch goes out of support, there are some things to garbage-collect.

- Edit /a/portbuild/admin/conf/admin.conf with the following changes:
  - Delete *old-branch* from SRC_BRANCHES .
  - Delete SRC_BRANCH_ *old-branch* _SUBDIR = *whatever*
- ```
  umount a/snap/src-old-branch /src;
  umount a/snap/src-old-branch ;
  zfs destroy --r a/snap/src-old-branch
  ```
- You will probably find that the following files and symlinks in /a/portbuild/error-logs/ can be removed:
  - Files named *-old_branch-failure.html
  ```

- Files named `buildlogs_*-old_branch-*-logs.txt`
- Symlinks named `*-old_branch-previous*`
- Symlinks named `*-old_branch-latest*`

# 16. How to rebase on a supported FreeBSD branch

As of 2011, the philosophy of package building is to build packages based on *the earliest supported release* of each branch. e.g.: if on `RELENG-8`, the following releases are supported: 8.1, 8.2, 8.3; then `packages-8-stable` should be built from 8.1.

As releases go End-Of-Life (see chart), a full (not incremental!) package build should be done and uploaded.

The procedure is as follows:

- Edit `/a/portbuild/admin/conf/admin.conf` with the following changes:
  - Change the value of `SRC_BRANCH_` *branch* `_SUBDIR` to `releng/`*branch*`.`*N* where `N` is the newest 'oldest' release for that branch.
- Run `/a/portbuild/admin/scripts/updatesnap` manually.
- Run `dopackages` with `-nobuild`.
- Follow the setup procedure.
- Now you can run `dopackages` without `-nobuild`.

# 17. How to configure a new architecture

## 17.1. Steps necessary before qmanager is started

Note

The next steps are most easily done as user `portbuild`.

### Note

The following assumes you have already run `mkportbuild`.

- As the `portbuild` user, run

```
% -/a/portbuild/tools/addarch arch
```

- For each branch that will be supported, do the following:
  - Kick-start the build for the branch with

```
# build create arch branch
```

- If you are going to store your historical buildlogs and errorlogs on your head node's hard drive, you may skip this step. Otherwise:

  Create an external directory and link to it:

#### Example 9. Creating and linking an external archive directory

```
# mkdir -/dumpster/pointyhat/arch/archive
# ln --s -/dumpster/pointyhat/arch/archive archive
```

### Note

(Historical note that only applied to the original `pointyhat.FreeBSD.org` installation)

It is possible that `/dumpster/pointyhat` will not have enough space. In that case, create the archive directory as `/dumpster/pointyhat/arch/archive` and symlink to that.

- Populate `clients` as usual.

- Edit `portbuild.conf` from one of the ones for another architecture. `addarch` will have created a default one for you.
- Create customized `portbuild.machinename.conf` files as appropriate.
- If you need to create any tunnels:
  1. Make a private configuration directory:

     ```
     # mkdir -/a/portbuild/conf/arch
     ```

  2. In that directory: create any `dotunnel.*` scripts needed.

Note

As `srcbuild`:

- Add *arch* to `SUPPORTED_ARCHS` in `/a/portbuild/admin/conf/admin.conf` .
- Add the *arch* directory to `/a/portbuild/admin/scripts/zbackup` . (This is a hack and should go away.)
- Enable the appropriate *arch* entry for `/a/portbuild/scripts/dologs` to the portbuild `crontab` . (This is a hack and should go away.)

## 17.2. Steps necessary after qmanager is started

Note

Again as `srcbuild`:

- For each branch that will be supported, do the following:
  - Create `bindist.tar` .

# 18. How to configure a new head node (pointyhat instance)

## 18.1. Basic installation

1. Install FreeBSD.

2. Create a user to own the portbuild repository, such as `portbuild`. It should have the `'*'` password.

3. Similarly, create a user to own the administration functions and manage the svn repositories, such as `srcbuild`. It should have the `'*'` password.

4. Add the following to `/boot/loader.conf` :

```
console="vidconsole,comconsole"
```

5. You should run the cluster on UTC. If you have not set the clock to UTC:

```
# cp --p -/usr/share/zoneinfo/Etc/UTC -/etc/localtime
```

6. Create the appropriate `/etc/rc.conf` .

   Required entries:

```
hostname="${hostname} "
sshd_enable="YES"
zfs_enable="YES"
```

   Recommended entries:

```
background_fsck="NO"
clear_tmp_enable="YES"
dumpdev="AUTO"
fsck_y_enable="YES"

apache22_enable="YES"
apache_flags=""
apache_pidfile="/var/run/httpd.pid"
inetd_enable="YES"
inetd_flags="-l --w"
mountd_enable="YES"
nfs_server_enable="YES"
nfs_server_flags="-u --t --n 12"
nfs_remote_port_only="YES"
ntpd_enable="YES"
rpcbind_enable="YES"
rpc_lockd_enable="NO"
rpc_statd_enable="YES"
sendmail_enable="NONE"
smartd_enable="YES"
```

If you are using ganglia, add:

```
gmetad_enable="YES"
gmond_enable="YES"
```

If you will be using a squid cache on the server, rather than the clients:

```
squid_enable="YES"
```

7. Create `/etc/resolv.conf` , if necessary.

8. Create the appropriate files in `/etc/ssh/` .

9. Add the following to `/etc/sysctl.conf` :

```
kern.maxfiles=40000
kern.maxfilesperproc=38000
sysctl vfs.usermount=1
sysctl vfs.zfs.super_owner=1
```

10. Make sure the following change is made to `/etc/ttys` :

```
ttyu0   -"/usr/libexec/getty std.9600"   vt100   on secure
```

## 18.2. Configuring `src`

You should be able to install from the most recent release using only the default kernel configuration.

## 18.3. Configuring `ports`

- The following ports (or their latest successors) are required:

```
databases/py-sqlite3
databases/py-sqlalchemy (only SQLITE is needed)
devel/git (WITH_SVN)
devel/py-configobj
devel/py-setuptools
devel/subversion
net/nc
net/rsync
www/apache22 (with EXT_FILTER)
```

Expect those to bring in, among others:

```
databases/sqlite3
lang/perl-5.14 (or successor)
lang/python27 (or sucessor)
```

If you are using ganglia, add:

```
sysutils/ganglia-monitor-core (with GMETAD off)
sysutils/ganglia-webfrontend (compile with --DWITHOUT_X11)
```

If you will be using a squid cache on the server, rather than the clients:

```
www/squid (with SQUID_AUFS on)
```

The following ports (or their latest successors) are strongly suggested:

```
devel/ccache
mail/postfix
net/isc-dhcp41-server
ports-mgmt/pkg
ports-mgmt/portaudit
ports-mgmt/portmaster
shells/bash
shells/zsh
sysutils/screen
```

Note

The use of sudo on the master, which was formerly required, is *no longer recommended*.

The following ports (or their latest successors) are handy:

```
benchmarks/bonnie++
ports-mgmt/pkg_tree
sysutils/dmidecode
sysutils/smartmontools
sysutils/zfs-stats
```

## 18.4. Configuring the zfs volume and setting up the repository

The following steps need to be done as euid root.

Here is a quick example:

Example 10. The contents of example file **portbuild/tools/example_install**

```
#!/bin/sh
#
```

```
# example script to drive the -"mkportbuild" kickstart file
#
export PORTBUILD_USER=portbuild
export SRCBUILD_USER=srcbuild
export ZFS_VOLUME=a
export ZFS_MOUNTPOINT=/a
export VCS_REPOSITORY=svn://svn0.us-east.FreeBSD.org

#
# create the zpool.  the examples here are just suggestions ↺
and need to be
# customized for your site.
#
# simple examples:
# zpool create ${ZFS_VOLUME} da1
# zpool create ${ZFS_VOLUME} gprootfs
# more complex example:
# zpool create ${ZFS_VOLUME} mirror da1 da2 mirror da3 da4 ↺
mirror da5 da6 mirror da7 da8

#
# check out the kickstart file and run it
#
mkdir --p tmp
svn checkout ${VCS_REPOSITORY}/base/projects/portbuild/admin/
tools tmp
sh --x -./tmp/mkportbuild
```

Here is a detailed explanation of the example:

1. Export the value of `PORTBUILD_USER` :

   ```
   # export PORTBUILD_USER=portbuild
   ```

2. Export the value of `SRCBUILD_USER` :

   ```
   # export SRCBUILD_USER=srcbuild
   ```

3. Pick a zfs volume name and export it. We have used *a* so far to date.

   ```
   # export ZFS_VOLUME=a
   ```

4. Pick a mountpoint and export it. We have used `/a` so far to date.

   ```
   # export ZFS_MOUNTPOINT=/a
   ```

5. Create the zfs volume and mount it.

### Example 11. Creating a zfs volume for portbuild

```
# zpool create ${ZFS_VOLUME} mirror da1 da2 mirror da3 ↺
da4 mirror da5 da6 mirror da7 da8
```

### Note

The kickstart script defines zfs `permission sets`, so that the *`srcbuild`* user and *`portbuild`* user may administer subdirectories of this volume without having to have root privileges.

6. Select an svn repository and export it. See the FreeBSD Handbook for the currently supported list.

```
# export VCS_REPOSITORY=svn://svn0.us-east.FreeBSD.org
```

7. Obtain a copy of the kickstart script into a temporary directory. (You will not need to keep this directory later.)

```
# mkdir --p -/home/portbuild /tmp
# svn checkout ${VCS_REPOSITORY}/base/projects/portbuild/admin/
tools -/home/portbuild /tmp
```

8. Run the kickstart script:

```
# sh -/home/portbuild /tmp/mkportbuild
```

This will accomplish all the following steps:

1. Create the `portbuild` directory
2. Create and mount a new zfs filesystem on it
3. Set up the directory
4. Set up the initial repository:
5. Set up the zfs `permission sets`.

6. Split ownerships of subdirectories such that *PORTBUILD_USER* owns, and only owns, files that are used to manage builds and interact with slaves. The more trustable user *SRCBUILD_USER* now owns everything else.

## 18.5. Configuring the srcbuild-owned files

- Configure the server by making the following changes to `/a/portbuild/admin/conf/admin.conf`:
  - Set `SUPPORTED_ARCHS` to the list of architectures you wish to build packages for.
  - For each source branch you will be building for, set `SRC_BRANCHES` and `SRC_BRANCH_`*branch*`_SUBDIR` as detailed in Section 14.1, "Steps necessary before qmanager is started". You should not need to change `SRC_BRANCHES_PATTERN`.
  - Set `ZFS_VOLUME` and `ZFS_MOUNTPOINT` to whatever you chose above.
  - Set `VCS_REPOSITORY` to whatever you chose above.
  - Set `MASTER_URL` to the http URL of your server. This will be stamped into the package build logs and the indices thereof.

  Most of the other default values should be fine.

## 18.6. Configuring the portbuild-owned files

1. Configure how build slaves will talk to your server by making the following changes to `/a/portbuild/conf/client.conf`:
   - Set `CLIENT_NFS_MASTER` to wherever your build slaves will PXE boot from. (Possibly, the hostname of your server.)
   - Set `CLIENT_BACKUP_FTP_SITE` to a backup site for FTP fetches; again, possibly the hostname of your server.
   - Set `CLIENT_UPLOAD_HOST` to where completed packages will be uploaded.

   Most of the other default values should be fine.
2. Most of the default values in `/a/portbuild/conf/common.conf` should be fine. This file holds definitions used by both the server and all its clients.
3. Configure the server by making the following changes to `/a/portbuild/conf/server.conf`:
   - Set `UPLOAD_DIRECTORY`, `UPLOAD_TARGET`, and `UPLOAD_USER` as appropriate for your site.

Most of the other default values should be fine.

## 18.7. pre-qmanager

- For each architecture, follow the steps in Section 17.1, "Steps necessary before qmanager is started".

## 18.8. qmanager

1. As `root`, copy the following files from `/a/portbuild/admin/etc/rc.d/` to `/usr/local/etc/rc.d/`:

```
pollmachine
qmanager
```

   As root, start each one of them. You may find it handy to start each under screen for debugging purposes.

2. Initialize the qmanager database's acl list:

> **Note**
>
> This should now be automatically done for you by the first `build` command.

```
# python -/a/portbuild/qmanager/qclient add_acl name=deny_all ↺
uidlist= gidlist= sense=0
```

## 18.9. Creating src and ports repositories

- As the *`srcbuild`* user, run the following commands manually to create the `src` and `ports` repositories, respectively:

```
% -/a/portbuild/admin/scripts/updatesnap.ports
% -/a/portbuild/admin/scripts/updatesnap
```

   These will be periodically run from the *`srcbuild`* `crontab`, which you will install below.

## 18.10. Other services

1. Configure `/usr/local/etc/apache22/httpd.conf` as appropriate for your site.

2. Copy `/a/portbuild/admin/conf/apache.conf` to the appropriate `Includes/` subdirectory, e.g., `/usr/local/etc/apache22/Includes/portbuild.conf`. Configure it as appropriate for your site.

3. Install `/a/portbuild/admin/crontabs/portbuild` as the `portbuild` crontab via `crontab -u portbuild -e`. If you do not support all the archs listed there, make sure to comment out the appropriate dologs entries.

4. Install `/a/portbuild/admin/crontabs/srcbuild` as the `srcbuild` crontab via `crontab -u srcbuild -e`.

5. If your build slaves will be pxebooted, make sure to enable the tftp entries in `/etc/inetd.conf`.

6. Configure mail by doing the following:

   `newaliases`.

## 18.11. Finishing up

1. For each architecture, follow the steps in Section 17.2, "Steps necessary after qmanager is started".

2. You will probably find it handy to append the following to the `PATH` definition for the *`portbuild`* user:

   ```
   /a/portbuild/scripts:/a/portbuild/tools
   ```

3. You will also probably find it handy to append the following to the `PATH` definition for the *`srcbuild`* user:

   ```
   /a/portbuild/admin/scripts:/a/portbuild/admin/tools
   ```

You should now be ready to build packages.

# 19. Procedures for dealing with disk failures

### Note

The following section is particular to `freebsd.org` and is somewhat obsolete.

When a machine has a disk failure (e.g., panics due to read errors, etc), then we should do the following steps:

- Note the time and failure mode (e.g., paste in the relevant console output) in `/a/portbuild/${arch}/reboots`

- For i386 gohan clients, scrub the disk by touching `/SCRUB` in the nfsroot (e.g., `/a/nfs/8.dir1/SCRUB` ) and rebooting. This will `dd if=/dev/zero of=/dev/ad0` and force the drive to remap any bad sectors it finds, if it has enough spares left. This is a temporary measure to extend the lifetime of a drive that is on the way out.

### Note

For the i386 blade systems another signal of a failing disk seems to be that the blade will completely hang and be unresponsive to either console break, or even NMI.

  For other build systems that do not newfs their disk at boot (e.g., amd64 systems) this step has to be skipped.

- If the problem recurs, then the disk is probably toast. Take the machine out of `mlist` and (for ata disks) run `smartctl` on the drive:

```
smartctl --t long -/dev/ad0
```

  It will take about 1/2 hour:

```
gohan51# smartctl --t long -/dev/ad0
smartctl version 5.38 [i386-portbld-freebsd8.0] Copyright (C) 2002-8
Bruce Allen
Home page is http://smartmontools.sourceforge.net/

=== START OF OFFLINE IMMEDIATE AND SELF-TEST SECTION ===
Sending command: -"Execute SMART Extended self-test routine ↺
immediately in off-line mode".
Drive command -"Execute SMART Extended self-test routine ↺
immediately in off-line mode" successful.
Testing has begun.
Please wait 31 minutes for test to complete.
Test will complete after Fri Jul  4 03:59:56 2008

Use smartctl --X to abort test.
```

  Then `smartctl -a /dev/ad0` shows the status after it finishes:

```
# SMART Self-test log structure revision number 1
# Num  Test_Description    Status                  Remaining
LifeTime(hours)  LBA_of_first_error
#   1  Extended offline    Completed: read failure       80%     ↺
15252    319286
```

It will also display other data including a log of previous drive errors. It is possible for the drive to show previous DMA errors without failing the self-test though (because of sector remapping).

When a disk has failed, please inform the cluster administrators so we can try to get it replaced.