Package Building Procedures

The FreeBSD Ports Management Team

Table of Contents

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 (http://www.FreeBSD.org/portmgr) team.

1.1 The codebase

Most of the package building magic occurs under the `/var/portbuild` directory. Unless otherwise specified, all paths will be relative to this location. *${arch}* will be used to specify one of the package architectures (amd64, i386™, ia64, powerpc, and SPARC64®), and *${branch}* will be used to specify the build branch (7, 7-exp, 8, 8-exp, 9, 9-exp, 10, 10-exp).

> **Note:** Packages are no longer built for Releases 4, 5, or 6, nor for the alpha architecture.

The scripts that control all of this live in `/var/portbuild/scripts/`. These are the checked-out copies from the Subversion repository `base/projects/portbuild/scripts/` (http://svnweb.freebsd.org/base/projects/portbuild/scripts/).

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 builds

1.2 Notes on the codebase

Until mid-2010, the scripts were completely specific to `pointyhat` 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

This document was originally written before these changes were made. Where things such as script invocations have changed, they are denoted as `new codebase:` as opposed to `old codebase:`.

> **Note:** As of December 2010, `pointyhat` is still running on the old codebase, until the new codebase is considered rock-solid.

> **Note:** Also during this process, the codebase was migrated to the Subversion repository (http://svnweb.freebsd.org/base/projects/portbuild/scripts/). For reference, the previous version may still be found in CVS (http://www.freebsd.org/cgi/cvsweb.cgi/ports/Tools/portbuild/scripts/Attic/).

2 Build Client Management

The i386 clients co-located with `pointyhat` netboot from it (`connected` nodes); all other clients (`disconnected` nodes) are either self-hosted or netboot from some other `pxe` host. In all cases they set themselves up at boot-time to prepare to build packages.

The cluster master `rsync`s 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 chroot builds.

The `ports-${arch}` 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.

The `scripts/checkmachines` script is used to monitor the load on all the nodes of the build cluster, and schedule which nodes build which ports. This script is not very robust, and has a tendency to die. It is best to start up this script on the build master (e.g. `pointyhat`) after boot time using a while(1) loop.

3 Chroot Build Environment Setup

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

The following command builds a world from the `${arch}/${branch}/`builds`/${buildid}/`src/ tree and installs it into `${worlddir}`. The tree will be updated first unless `-nocvs` is specified.

```
/var/portbuild# scripts/makeworld ${arch} ${branch} ${buildid} [-nocvs]
```

The `bindist.tar` tarball is created from the previously installed world by the `mkbindist` script. It should be run as `root` with the following command:

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

The per-machine tarballs are located 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.

3.1 New Codebase

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

4 Customizing Your Build

(The following only applies to the new codebase.)

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
# Don't 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

Several separate builds for each architecture - branch combination are supported. All data private to a build (ports tree, src tree, packages, distfiles, log files, bindist, Makefile, etc) are located under
${arch}*/*${branch}/builds/*${buildid}*. The last created build can be alternatively referenced under buildid latest, the one before is called previous.

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

5.1 `dopackages` scripts

The scripts/dopackages scripts are used to perform the builds.

5.1.1 Old codebase

For the old codebase: the most useful are:

* dopackages.7 - Perform a 7.X build

* dopackages.7-exp - Perform a 7.X build with experimental patches (7-exp branch)

* dopackages.8 - Perform a 8.X build

* dopackages.8-exp - Perform a 8.X build with experimental patches (8-exp branch)

* dopackages.9 - Perform a 9.X build

* dopackages.9-exp - Perform a 9.X build with experimental patches (9-exp branch)

* dopackages.10 - Perform a 10.X build

* dopackages.10-exp - Perform a 10.X build with experimental patches (10-exp branch)

These are wrappers around dopackages, and are all symlinked to dopackages.wrapper. New branch wrapper scripts can be created by symlinking dopackages.${branch} to dopackages.wrapper. These scripts take a number of arguments. For example:

```
dopackages.7 ${arch} ${buildid} [-options]
```

5.1.2 New codebase

The symlinks go away, and you just use `dopackages.wrapper` directly. For example:

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

5.1.3 Either codebase

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. Don't 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, don't forget to cleanup the clients when you don't need the build anymore.

- `-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. (New codebase only).

- `-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, on the new codebase, `-unlimited-errors`).

- `-nosrc` - Do not update the `src` tree from the ZFS snapshot, keep the tree from previous build instead.

- `-srccvs` - Do not update the `src` tree from the ZFS snapshot, update it with `cvs update` instead.

- `-noports` - Do not update the `ports` tree from the ZFS snapshot, keep the tree from previous build instead.

- `-portscvs` - Do not update the `ports` tree from the ZFS snapshot, update it with `cvs update` 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 `ftp-master`.

- `-unlimited-errors` (new codebase only) - 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
```

`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 `${arch}` build is run as the ports-`${arch}` 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}/make.[0|1]`:

```
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.7 i386 -nosrc -norestr -nofinish
dopackages.wrapper i386 7 -nosrc -norestr -nofinish
```

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

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

Example 8. Post-process a completed sparc64-7 tree

```
dopackages.7 sparc64 -finish
dopackages.wrapper sparc64 7 -finish
```

Hint: it us 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. (TODO: document whether newid must be specified as `latest` in the new codebase.)

- `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. Don't forget to use `-nosrc` flag to `dopackages` later!

- `build portsupdate` `arch branch buildid` - Replaces the ports tree with a new ZFS snapshot. Don't 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 [*]

2. An update of the running branch's `src` tree from the ZFS snapshot [*]

3. Checks which ports do not have a `SUBDIR` entry in their respective category's `Makefile` [*]

4. Creates the `duds` file, which is a list of ports not to build [*] [+]

5. Generates a fresh `INDEX` file [*] [+]

6. Sets up the nodes that will be used in the build [*] [+]

7. Builds a list of restricted ports [*] [+]

8. Builds packages (phase 1) [++]

9. Performs another node setup [+]

10. Builds packages (phase 2) [++]

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

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

[++] Status of these steps can be found in `${arch}/${branch}/make` (old codebase) or `${arch}/${branch}/journal` (new codebase). Individual ports will write their build logs to `${arch}/${branch}/logs` and their error logs to `${arch}/${branch}/errors`.

Formerly the docs tree was also checked out, however, it has been found to be unnecessary.

7 Build Maintenance

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

1. You have manually interrupted it.

2. `pointyhat` 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 `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:

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

If you forget to do this, then the old build `chroot`s 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's less than the number of jobs that `loads` thinks are in use, you're in trouble.

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 -l root 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 2 mean that that client did not have those mounted; the latter 2 are a bug.
>
> You may also see messages about `procfs`.

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.

- `build upload` *`arch branch`* - not yet implemented.

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 /var/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 `/var/portbuild` file system becomes full then Bad Things™ happen.

The status of all current builds is generated twice an hour and posted to http://pointyhat.FreeBSD.org/errorlogs/packagestats.html. For each `buildenv`, the following is displayed:

- `cvs date` is the contents of `cvsdone`. This is why we recommend that you update `cvsdone` 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 `cvs 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 `-nocvs` and `-noportscvs`.

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}
```

This inventory file typically lives in `i386/${branch}` on the cluster master.

This is useful to aid in periodically cleaning 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

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 /var/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:

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

Doing it by hand is deprecated.

12 Experimental Patches Builds

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:

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

This will most closely simulate what the `dopackages` script does. (While `cvsdone` 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 /var/portbuild/i386/8-exp/errors
% find . -name \*.log\* | sort > /tmp/8-exp-errs
% cd /var/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 [1]

- Port was upgraded since the control build and has become broken [2]

- 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 [1] and [2] above, you can do a rebuild of the affected packages under the control branch:

% **cd /var/portbuild/i386/8/ports**

> **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 (old codebase):

% **/var/portbuild/scripts/dopackages.8 –noportscvs –nobuild –nocvs –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 /var/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 `/var/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 (mailto:ports@FreeBSD.org) and copy ports-developers@FreeBSD.org (mailto: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`.

> **Note:** Due to some generous donations, `portmgr` is no longer looking for the loan of i386 or `amd64` systems. However, we are still interested in borrowing tier-2 systems.

13.1 Node requirements

`portmgr` is still working on characterizing 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/`.)

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 you are using more than 3.5G on i386, 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


   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 `ports-`*${arch}* user and group. Add it to the `wheel` group. It can have the '`*`' password.

  Create `/home/ports-`*${arch}*`/.ssh/` and populate `authorized_keys`.

- Also add the following users:

  ```
  squid:*:100:100::0:0:User &:/usr/local/squid:/bin/sh
  ganglia:*:102:102::0:0:User &:/usr/local/ganglia:/bin/sh
  ```

  Add them 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
  ```

- We run the cluster on UTC:

  ```
  cp /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"
  ntpdate_enable="YES"
  ntpdate_flags="north-america.pool.ntp.org"
  sendmail_enable="NONE"
  sshd_enable="YES"
  sshd_program="/usr/local/sbin/sshd"

  gmond_enable="YES"
  squid_enable="YES"
  squid_chdir="/usr2/squid/logs"
  squid_pidfile="/usr2/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="/usr2/squid/logs"
squid_pidfile="/usr2/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 `/usr2` persistent (which should save instantiation time.) Work is still ongoing.

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

- Modify `etc/sysctl.conf`:

```
9a10,30
> kern.corefile=/usr2/%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 don't 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/openssh-portable (with HPN on)
    security/sudo
    sysutils/ganglia-monitor-core (with GMETAD off)
    www/squid (with SQUID_AUFS on)
    ```

    There is a WIP to create a meta-port, but it is not yet complete.

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:

    *   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=usr2

    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
```

- 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`.

- Modify `usr/local/etc/sudoers`:

```
38a39,42
>
> # local changes for package building
> %wheel         ALL=(ALL) ALL
> ports-${arch}   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 ports-${arch}:ports-${arch} portbuild
mkdir pkgbuild
chown ports-${arch}:ports-${arch} pkgbuild
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 chroots from previous builds before creating `/tmp/.boot_finished`.

4. Boot the client.

5. As root, initialize the `squid` directories:

   squid -z

13.6 Configuration on `pointyhat`

These steps need to be taken by a `portmgr` acting as `ports-${arch}` on `pointyhat`.

1. If any of the default TCP ports is not available (see above), you will need to create an `ssh` tunnel for it and include it in the `crontab`.

2. Add an entry to `/home/ports-${arch}/.ssh/config` to specify the public IP address, TCP port for `ssh`, username, and any other necessary information.

3. Create `/var/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 `/var/portbuild/conf/clients/`) and symlink to it.

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

   Suggested values:

   ```
   disconnected=1
   http_proxy="http://localhost:3128/"
   squid_dir=/usr2/squid
   scratchdir=/usr2/pkgbuild
   client_user=ports-${arch}
   sudo_cmd="sudo -H"
   rsync_gzip=-z

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

   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. 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 `ports-arch` on `pointyhat`.

1. Ensure that `ssh` is working by executing `ssh hostname`.

2. Populate `/var/portbuild/scripts/` by something like `/var/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 `root` on `pointyhat`.

1. 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"
   ```

14 How to configure a new FreeBSD branch

When a new branch is created, some work needs to be done to specify that the previous branch is no longer equivalent to HEAD. The following instructions apply to the *previous* branch number:

- (new codebase) Edit `/var/portbuild/conf/server.conf` with the following changes:
  - Add *new-branch* to `SRC_BRANCHES`.
  - For what was previously head, change `SRC_BRANCH_branch_TAG` to `RELENG_branch_0`.
  - Add `SRC_BRANCH_new-branch_TAG = .` (literal period).

- (new codebase) Run `/var/portbuild/updatesnap` manually.

- (Only for old codebase) Create a new **zfs** filesystem for sources:

  `zfs create a/snap/src-`*branch*

- (Only necessary for old codebase): Checkout a `src` tree in the new filesystem:

  `cvs -Rq -d /r/ncvs co -d src-`*branch*`-r RELENG_`*branch*

- (Only necessary for old codebase): Edit the master copy of `Tools/portbuild/portbuild.conf`.

- (Only necessary for old codebase): For each arch, edit its copy of the above in
  `/var/portbuild/`*arch*`/portbuild.conf`.

- (Only necessary for old codebase): Edit `/var/portbuild/scripts/buildenv`.

- (Only necessary for old codebase): Add a link from `/var/portbuild/scripts/dopackages` to
  `/var/portbuild/scripts/dopackages.`*branch*.

- (Only necessary for old codebase): Modify `HEAD_BRANCH` and `NON_HEAD_BRANCHES` in
  `/var/portbuild/scripts/updatesnap`.

- (Only necessary for old codebase): Add the `snap` directory to `/var/portbuild/scripts/zexpire`.

- (Only necessary for old codebase): In the `/var/portbuild/errorlogs/` directory, create links for the
  webserver:

  ```
  ln -s ../arch/branch/builds/latest/bak/errors arch-branch-full
  ln -s ../arch/branch/builds/latest/bak/logs arch-branch-full-logs
  ln -s ../arch/branch/builds/latest/errors arch-branch-latest
  ln -s ../arch/branch/builds/latest/logs arch-branch-latest-logs
  ln -s ../arch/branch/builds/latest/bak/packages arch-branch-packages-full
  ln -s ../arch/branch/builds/latest/packages arch-branch-packages-latest
  ```

- Kick-start the build for the branch with

  `build create `*arch branch*

- 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.

- (new codebase) Edit `/var/portbuild/conf/server.conf` with the following changes:

  - Delete *old-branch* from `SRC_BRANCHES`.

  - Delete `SRC_BRANCH_`*old-branch*`_TAG =`*whatever*

- (both): `umount a/snap/src-`*old-branch*`/src; umount a/snap/src-`*old-branch*`; zfs destroy -r
  a/snap/src-`*old-branch*

- (both) You will probably find that the following files and symlinks in `/var/portbuild/errorlogs/` 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 (http://www.freebsd.org/security/index.html#supported-branches)), a full (not incremental!) package build should be done and uploaded.

The procedure for the new codebase is as follows:

- Edit `/var/portbuild/conf/server.conf` with the following changes:
  - Change `SRC_BRANCH_`*`branch`*`_TAG` to `RELENG_`*`branch`*`_N` where `N` is the newest 'oldest' release for that branch.

- Run `/var/portbuild/updatesnap` manually.
- Run `dopackages` with `-nobuild`.
- Follow the setup procedure.
- Now you can run `dopackages` without `-nobuild`.

The procedure for the old codebase is left as an exercise for the reader.

17 How to configure a new architecture

> **Note:** The initial steps need to be done using **sudo**.

- Create a new `ports-`*`arch`* user and group.
- `mkdir /var/portbuild/`*`arch`*
- Create a new **zfs** filesystem:

  `zfs create -o mountpoint=/a/portbuild/`*`arch`* `a/portbuild/`*`arch`*

- `chown ports-`*`arch`*`:portmgr /var/portbuild/`*`arch`*`;`
`chmod 755 /var/portbuild/`*`arch`*`;`
`cd /var/portbuild/`*`arch`*
- Create and populate the `.ssh` directory.
- Create a directory for buildlogs and errorlogs:

```
mkdir /dumpster/pointyhat/arch/archive
```

> **Note:** 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. (This needs to be sorted out.)

- Create a link to the above for the webserver:

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

> **Note:** The next steps are most easily done as user `ports-arch`.

- In the `/var/portbuild/arch` directory:

```
mkdir clients
```

- Populate `clients` as usual.

-
```
mkdir loads
```

-
```
mkdir lockfiles
```

- Create a local `make.conf`. In the most trivial case, you can

```
ln ../make.conf ./make.conf
```

- Create an empty `mlist` file.

- (Only necessary for old codebase) Create `pnohang.arch`. (The easiest way may be to do the following on a client, and then copy it back):

```
cc pnohang.c -o pnohang-arch
```

- Create a fresh `portbuild.conf` file from one of the ones for another architecture.

- Create customized `portbuild.machinename.conf` files as appropriate.

-
```
cd .ssh && ssh-keygen
```

- Edit the `.ssh/config` file for convenience in using **ssh**.

- Make the private configuration directory:

```
mkdir /var/portbuild/conf/arch
```

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

> **Note:** Once again using **sudo**:

- Tell **qmanager** about the arch:

  ```
  python path/qmanager/qclient add_acl name=ports-arch uidlist=ports-arch gidlist=portmgr sense=1
  ```

- (Only necessary for new codebase): Add *arch* to SUPPORTED_ARCHS in
  /var/portbuild/*arch*/server.conf.

- (Only necessary for old codebase): Edit /var/portbuild/scripts/buildenv.

- Add the *arch* directory to /var/portbuild/scripts/zbackup and /var/portbuild/scripts/zexpire.

- (Only necessary for old codebase): As with the procedure for creating a new branch: in the
  /var/portbuild/errorlogs/ directory, create links for the webserver:

  ```
  ln -s ../arch/branch/builds/latest/bak/errors arch-branch-full
  ln -s ../arch/branch/builds/latest/bak/logs arch-branch-full-logs
  ln -s ../arch/branch/builds/latest/errors arch-branch-latest
  ln -s ../arch/branch/builds/latest/logs arch-branch-latest-logs
  ln -s ../arch/branch/builds/latest/bak/packages arch-branch-packages-full
  ln -s ../arch/branch/builds/latest/packages arch-branch-packages-latest
  ```

- In the /var/portbuild/errorlogs/ directory, create two more links for the webserver:

  ```
  ln -s ../arch/archive/buildlogs arch-buildlogs
  ln -s ../arch/archive/errorlogs arch-errorlogs
  ```

  Note: Again, as ports-arch:

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

    ```
    build create arch branch
    ```
  - Create bindist.tar.

    Note: One last time using **sudo**:

- (Only necessary for old codebase): Only after the first time a **dopackages** has been run for the arch: add the arch
  to /var/portbuild/scripts/dopackagestats.

- Add an appropriate *arch* entry for /var/portbuild/scripts/dologs to the root crontab. (This is a hack
  and should go away.)

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

This section is in progress.

Please talk to Mark Linimon before making any changes.

18.1 Basic installation

1. Install FreeBSD.

2. For each supported arch, add a `ports-${arch}` user and group. Add them to the `wheel` group. They should have the '`*`' password. Also, similarly, create the `ports` and `portmgr` users.

3. For each supported arch, create `/home/ports-${arch}/.ssh/` and populate `authorized_keys`.

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

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

   ```
   kern.maxfiles=40000
   ```

6. TBA

18.2 Configuring the disk

1. Create a **zfs** volume named `a` and mount it on `/a`.

2.

   ```
   # mkdir -p /a/portbuild
   # cd /a/portbuild
   #
   # chmod 775 .
   ```

3. Create a **zfs** volume named `a/portbuild` and mount it on `/a/portbuild`.

4. TBA

18.3 Configuring `src`

1. TBA

18.4 Configuring `ports`

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

   ```
   databases/py-pysqlite23
   databases/py-sqlalchemy
   devel/git (WITH_SVN)
   devel/py-configobj
   ```

```
devel/py-setuptools
devel/subversion
net/nc
net/rsync
sysutils/ganglia-monitor-core (with GMETAD off)
sysutils/ganglia-webfrontend (WITHOUT_X11)
www/apache22 (with EXT_FILTER and THREADS)
```

Expect those to bring in:

```
databases/sqlite3
lang/perl-5.12
lang/python27
```

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

```
benchmarks/bonnie++
devel/ccache
mail/postfix
net/isc-dhcp41-server
ports-mgmt/pkg_cutleaves
ports-mgmt/pkg_tree
ports-mgmt/portaudit
ports-mgmt/portmaster
security/sudo
shells/bash
shells/zsh
sysutils/screen
sysutils/smartmontools
```

2. Configure mail by doing the following: TBA.

18.5 Other

1. TBA

19 Procedures for dealing with disk failures

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
  /var/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 don't 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.