Package Building Procedures

The FreeBSD Ports Management Team

$FreeBSD: doc/en_US.ISO8859-1/articles/portbuild/article.sgml,v 1.36 2009/01/07 09:34:58 pav Exp $

Table of Contents

1 Introduction and Conventions

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.

Most of the package building magic occurs under the `/var/portbuild` directory. Unless otherwise specified, all paths will be relative to this location. *`${arch}`* will be used to specify one of the package architectures (amd64, i386™, and Sparc64®), and *`${branch}`* will be used to specify the build branch (6, 6-exp, 7, 7-exp, 8, 8-exp).

> **Note:** Packages are no longer built for Release 4 or 5, nor for the alpha nor ia64 architectures.

The scripts that control all of this live in `/var/portbuild/scripts/`. These are the checked-out copies from `/usr/ports/Tools/portbuild/scripts/`.

Typically, incremental builds are done that use previous packages as dependendencies; 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
- every month or so, for `-CURRENT`
- for experimental builds

2 Build Client Management

The i386 clients currently netboot from `pointyhat`; the other clients are self-hosted. In all cases they set themselves up at boot-time to prepare to build packages.

Although connected nodes are supported, *`disconnected`* cluster node support has been added. A disconnected node is one that does not mount the cluster master via NFS. It could be a remote node, for example. 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}`*`/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.

4 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.

4.1 dopackages scripts

The scripts/dopackages* scripts are used to perform the builds. Most useful are:

- dopackages.6 - Perform a 6.X build
- dopackages.6-exp - Perform a 6.X build with experimental patches (6-exp branch)
- 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)

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.6 ${arch} [-options]
```

[-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-finished) 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.
- `-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).
- `-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.
- `-plistcheck` - 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`.

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 1. Update the i386-6 tree and do a complete build

```
dopackages.6 i386 -nosrc -norestr -nofinish
```

Example 2. Restart an interrupted amd64-8 build without updating

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

Example 3. Post-process a completed sparc64-7 tree

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

4.2 `build` command

You may need to manipulate the build data before starting it, especially for experimental builds. This is done with `build` command.

- `build list` *`arch branch`* - Shows the current set of build ids.
- `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!

4.3 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:

```
/var/portbuild/evil/qmanager/packagebuild amd64 7-exp 20080904212103 aclock-0.2.3_2
```

5 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.[0|1]`, where `make.0` is the log file used by phase 1 of the package build and `make.1` is the log file used by phase 2. 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.

6 Interrupting a Build

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 p1`; otherwise issue a `kill -HUP` in there by, for example, `ps -t p1 -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.

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 6-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/i386/6-exp/ports"
allgohans "umount -f pointyhat.freebsd.org:/var/portbuild/i386/6-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/i386/6-exp/ports: statfs: No such file or directory
> umount: pointyhat.freebsd.org:/var/portbuild/i386/6-exp/ports: unknown file system
> umount: Cleanup of /x/tmp/6-exp/chroot/53837/compat/linux/proc failed!
> /x/tmp/6-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 `cvs update` before restarting, you may have to rebuild either `duds`, `INDEX`, or both. If you are doing the latter manually, you will also have to rebuild `packages/All/Makefile` via the `makeparallel` script.

7 Monitoring the Build

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

```
python /var/portbuild/evil/qmanager/qclient jobs
```

```
python /var/portbuild/evil/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`.

8 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`.

9 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.

10 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-6.3-release`
- `packages-6-stable`
- `packages-7.0-release`
- `packages-7-stable`
- `packages-8-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 can be transferred with the `cpdistfiles` script:

```
# /var/portbuild/scripts/cpdistfiles ${arch} ${branch}
```

Or you can do it by hand using `rsync` command:

```
# cd /var/portbuild/${arch}/${branch}
# rsync -n -r -v -l -p -c distfiles/ portmgr@ftp-master:w/ports/distfiles/ | tee log
```

Again, run the command without the `-n` option after you have checked it.

11 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. The current experimental patches branch is `7-exp` on the i386 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 `6`). 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.

> **Note:** One build cluster can do the control build while the other does the experimental patches build. This can be a great time-saver.

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 `6` branch is the control branch, and the `6-exp` branch is the experimental patches branch):

```
% cd /var/portbuild/i386/6-exp/errors
% find . -name \*.log\* | sort > /tmp/6-exp-errs
% cd /var/portbuild/i386/6/errors
% find . -name \*.log\* | sort > /tmp/6-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/6-errs /tmp/6-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/i386/6/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:

```
% /var/portbuild/scripts/dopackages.6 -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/i386/6/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`.

12 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 Kris Kennaway <`kris@FreeBSD.org`> so he can try to get it replaced.