Lightweight FreeBSD package cluster in a jail
Version 2004-09-22

Dirk Meyer
dinoex@freebsd.org
http://people.freebsd.org/~dinoex/

How to setup a `jail' to ensure clean package builds on FreeBSD, making it easy to distribute customized packages for more than one machine. Also, how to use this scripts as a test environment to compile new or modified ports.

1. Generation of the packages

1.1. Motivation to start this project

When the ports I maintained became more complex, I needed a clean environment to test existing and new ports.

1.2. Differences to the `bento' cluster system

A. It works in a `perl' free environment

FreeBSD packages are built on the `bento' or `pointyhat' cluster. The infrastructure for this is designed for parallel package building on more than two machines. It makes heavy use of `perl' scripts, which was the main reason I have dropped the idea of using this as a base for testing my own ports.

I had many problems with `perl' scripts. The transition from `perl4' to `perl5' was difficult, with many portability problems and script incompatibility.

B. It runs inside or outside a `jail'

Running `chroot' as on the cluster will not protect your applications on the host. Some ports try to be very smart by stopping services while programs install or uninstall, e.g. the `cups' port still kills all running `cupsd' processes. By running in a `jail', this and other side effects can be prevented.

I didn't want to endanger my base machine, and I wanted to get rid of the many packages that are installed as `build only' dependencies. In a clean `jail', I don't have to worry about a lot of side effects, such as auto detection of extra installed libs, which quickly become an unexpected dependency. For example, the `samba' port sucked in the `popt' lib without registering it as dependency. So, when I removed `rpm' and the unused `popt' from the system, `samba' refused to start. This problem has been fixed already, but there are several similar pitfalls.

You can start one `jail' IP-address for only fetching the ports you need, and use a different `jail' IP-address for the build. This allows you to block any access from your build jail in your firewall, giving you additional protection from Trojan programs.

C. KISS - Keep it small and simple

Only the tools in the base system, and `cvsup', are used. I use `make' to extract all vital information, and use `shell' to get everything working together. For processing the `plist', I used a few lines of `awk'.

It is designed for a single jail building all packages you need. Then, you can install the ready made packages quickly on every machine. One `nfs mount' of `/usr/ports' contains all files you need to install or upgrade a system. This is the reason why I chose `/usr/ports/local/update/' as the default location for my tools.

1.3. `INDEX' problems, Customizing

In the FreeBSD ports tree, the `INDEX' files are very large, caching a lot of information from 11739 ports (FreeBSD 5.3 BETA). It weights about 5.5M for each branch. These files are only up to date when a RELEASE is generated, and in the meantime, you have to rebuild the INDEX each time a `Makefile' in the ports tree changes. A lot of package tools use these files, but the information may not be consistent with the environment the user has chosen. Dependencies may be changed at any time by options and settings in `/etc/make.conf'. So you waste a lot of effort rebuilding the `INDEX' files and keeping the information in there up to date. If using the target `fetchindex', you get a snapshot that is more recent, but you can't trust that the information in there is still correct.

Instead, I save bandwidth by ignoring all `INDEX' files and rely only on the current information provided by the ports `Makefiles'. The trade off in computing time is smaller than I estimated, because only the needed ports are processed, but the lack of caching shows on the big dependency trees like in `gnome' or `kde' and will slow you down again.

Parts of my ideas have been discussed on the ´freebsd-ports' mailing list, and I believe in avoiding the `INDEX' files at all will be a step in the right direction.

1.4. Preserving good builds and reusing them

The FreeBSD package cluster builds all ports on each run. Having limited amount of computer power, I decided to reuse packages untill they become obsolete. Therefore, each dependency is built as a package, so it can be reused for new builds of a port and for other ports depending on it.

The current ports system supports this too by setting `/etc/make.conf'.
I recommend the following settings for each host you want to install packages on.

```
DEPENDS_TARGET=package
USE_PACKAGE_DEPENDS=yes
```

1.5. How to decide if a port needs to be rebuilt

Instead of using the package name, I decided to use only one key to represent a port or dependency: the path to the port's directory. This allows fast reference to the originating `Makefile' and access to all the current information we need.

Step 1.
I determine the affected ports by path name. This is easy because my scripts can use the standard target `all-depends-list'. There is no need to distinguish between build, lib and run depends, as building the package needs them all.

```
root@jail:/usr/ports/archivers/arj# make all-depends-list
/usr/ports/converters/libiconv
/usr/ports/devel/autoconf253
/usr/ports/devel/gettext
/usr/ports/devel/gmake
/usr/ports/devel/libtool13
/usr/ports/devel/libtool15
/usr/ports/devel/m4
/usr/ports/devel/p5-Locale-gettext
/usr/ports/misc/help2man
/usr/ports/textproc/expat2
```

Step 2.
I check each dependency for the exact version. I get this by using the target `package-name'. If the exact version is not installed, we have to add or build the dependency. If a required package is newer than an already existing package of the port we want, a rebuild is triggered. This is very useful to catch any changes in the `build only' dependency.

```
root@jail:/usr/ports/devel/autoconf253# make package-name
autoconf-2.53_3
```

1.6. Build only the packages you need

Several operation modes are supported. When you call the script with the directory of a port, it will only check and build the port and its dependencies. This can be used for emergency patches and for simply testing a new port.

Calling the script with a file will process the directories listed in that file for rebuild. This is useful to build a couple of related ports.

Finally, if you omit the file, then a stored host specific list will be used.

1.7. Performance in FreeBSD4 vs. FreeBSD5

Running the `jail' on FreeBSD 4.x is about 4 times faster than with FreeBSD 5.x. There are two reasons for this: the `bzip2' compressed packages takes much longer to access or extract, and the `gcc3.4' compiler takes up much more time. You can regain a bit of time by reverting the compression scheme for the packages to `gzip'.

1.8. Caveats: e.g. Linux emulations

There are some ports that won't build in the `jail'. The `linux' emulation is a prominent example of this. In this case, you can build it outside with `chroot', and the resulting package can be used to build dependent ports in the `jail' again, until it becomes obsolete.

2. Keeping up to date

2.1. Installation and Layout

You extend your ports tree by creating the directory:

```
root@host# mkdir -p /usr/ports/local/update
```

Download the files from `http://people.freebsd.org/~dinoex/batch/`
and don't forget the `README'.

You may add extra ports under `/usr/ports/local', I create a slave port there when I need more than one package from the same port.

Inside the `jail' I use the minimal `/etc/make.conf':

```
USA_RESIDENT=NO
WRKDIRPREFIX=/image
PACKAGES=/usr/ports/packages
BATCH=yes
SUP_UPDATE?=yes
SUP?=/usr/local/bin/cvsup
SUPFLAGS?=-g -L 2
SUPHOST?=cvsup.de.FreeBSD.org
PORTSSUPFILE?=/usr/share/examples/cvsup/ports-supfile
```

2.2. Create a list of installed packages

You can create a starting configuration easy like this:

```
root@host# pkg_info -qao > /usr/ports/local/update/\
data/make-packages.hostname
```

This is sufficient, but you may optimize the order of ports to match your needs. Feel free to remove dependencies, so they won't be built once they are no longer in use, or put them in any order you prefer.

2.3. Stetting up the jail

Extract a release in a directory of your choice, or use a current or stable build with `make DESTDIR=/jail5 installworld'. I recommend you keep the `jail' up to date with each `buildworld'. In the host system, you should enable the `IPC' for the `jail'. This is needed for some databases and for the `sendmail' ports. I set this in `/etc/sysctl.conf':

FreeBSD 4.x:
```
jail.sysvipc_allowed=1
```

FreeBSD 5.x:
```
security.jail.sysvipc_allowed=1
```

In FreeBSD 5.x you have to mount the `devfs' in the jail. Some ports may require the `prcofs' mounted as well, if you do this you should mount `procfs' read only, I use read only `procfs' on the hosting system as well.

2.4. Running each day

The update cycle is easy. It can be run unattended, so you have the new packages at hand when you decide to update. I run `cvsup' to update the ports tree, then I have to get rid of obsolete packages. After removing log files from aborted builds, I am ready to rebuild all the missing packages.

2.5. Find outdated packages

I have to check each package for its origin and the registered dependencies. In case we detect a difference, we move the package aside, so we have the latest package around in case the new build might fail.

2.6. Running through the dependency tree

The scripts are very picky about each version, older or newer don't matter. You can even do a clean `downdate' if necessary, because an exact matching version is enforced on each run.

2.7. How to clean your `distfiles' directory

Once in a while you like to clean up your `distfiles' directory. This was much easier than I thought. I look for each `distinfo' file in the whole ports tree and compare the list with the list of downloaded `distfiles'. Each file that is no longer listed in the `distinfo' in any port can be safely moved away.

2.8. Small and full upgrades

Each run will generate the newest packages you need. Mostly it will need less than half an hour, but when a major dependency has changed it can run up to more than one day.

If you upgrade your base system, I recommend you move all packages away. Some ports have paths that change with the FreeBSD version, or includes and libraries in the base change, and if you keep the old packages, other problems might occur.

2.9. Naming problems when ports using auto detection

Some ports change their package name while build, and this will be recorded as a failure, but the built package will kept around. Setting build options ahead for the port will give you a clean build.

To set options for a specific port when settings in `Makefile.local' are not working.
I was successful to place this as a conditional in `/etc/make.conf':

```
.if ${.CURDIR} == "/usr/ports/multimedia/mplayer"
WITH_GUI=yes
WITH_GTK1=yes
WITHOUT_ESOUND=yes
CFLAGS+=-O
.endif
```

When this is more than 3 lines, I strongly recommend to build with a simple slave port. I found an example for this `/usr/ports/local/mplayer-extra/Makefile':

```
PKGCATEGORY?=   local
MASTERDIR=      /usr/ports/multimedia/mplayer

WITH_GUI=yes
WITH_SDL=yes
WITH_VORBIS=yes
WITH_XANIM=yes
WITH_FREETYPE=yes

WITHOUT_RUNTIME_CPUDETECTION=yes
WITHOUT_3DNOW=yes

.include "${MASTERDIR}/Makefile"
```

2.10. Errors

Well in active development of the ports, logfiles are preserved in the log subdir. Looking at current sample you can see three different types of files.

```
-rw-r--r--  1 root  wheel     24890 Sep 15 18:13
build,local,gnumail
-rw-r--r--  1 root  wheel       120 Sep 15 18:14 plist,
local,gnumail
-rw-r--r--  1 root  wheel     14925 Sep 15 18:23 build,
local,projectcenter.app
-rw-r--r--  1 root  wheel     11309 Sep 15 18:37 build,
local,preferences.app
-rw-r--r--  1 root  wheel       161 Sep 15 18:38 plist,
local,preferences.app
-rw-r--r--  1 root  wheel     57388 Sep 15 18:56 build,
local,gworkspace
-rw-r--r--  1 root  wheel       198 Sep 15 18:57 plist,
local,gworkspace
-rw-r--r--  1 root  wheel      5290 Sep 16 06:25 err,
local,gnumail_112
```

First there are successful build logs from ports with a name as:
`build,<category>,<port>'. They stay around until the next successful build. In case of problems I found the `configure' output there is very helpful.

Second we may have files with a name such as: `plist,<category>,<port>'.
This indicates that after building this package and deleting its dependency, additional files or directories were found. Directories can be mostly ignored, but missing files can indicate a problem with the port or its dependency. If you are a maintainer, you can fix this by updating the `pkg-plist' of your port.

Last we have a log named `err,<category>,<port>'. This can be a build log in progress or an aborted build. In this case, a `diff' between the new log file and the last successful build log can be helpful to find the cause.

2.9. Improvements

This project was started in 2002, and has been tested at different locations. It's development is stable, the history can be reviewed via `cvsweb'.