Performing clean updates with FreeBSD packages

Target-Audience: Everybody

Target-OS: FreeBSD

Time schedule: Tutorial half day

Location: Eurobsdcon 2008, Thursday - October 16th 2008

Author: Dirk Meyer, FreeBSD user since 2.1.0, ports-committer since 2001

[dirk.meyer@dinoex.sub.org],[dirk.meyer@guug.de],[dinoex@FreeBSD.org]

Abstract

How-to manage your own pre build packages and use them as binary updates for distribution over different machines, reducing the time for updating and maintaining your systems, spanning from small servers to full grown desktops environments on your workstations. Either to maintain a stable state or keep a bleeding edge on the applications in use, while using packages in a reproduce able and consistent way.

Build your packages in a jail, and get rid off ‘build only’ dependencies on the machines your work with, and ensure the chain of dependencies is clean and in best order, e.g. using the lightweight package cluster.

Update strategies which are small updates, partial and full rebuild. Deinstallation packages in order, avoiding the evil -f. Reinstall packages in order and keep track of your installed software.

Customize your packages by creating local ports, which may forked of a normal port or making a slave port. Test builds and a quick walk through for creating new ports.

Overview

- 1. Setup of a jail
- 2. Ready to start
- 3. Ready to build
- 4. Updating the host
- 5. Extending your ports
- 6. References
- 7. Questions

1. Setup of a jail

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.

1.1. Installation of the clean jail.

Install a base system by extracting the ‘bin’ and ‘etc’ distribution in a directory. If you have your sources, and have run ‘make buildworld’, then you can install with make. For convenience I have a script for setup.

$ fetch http://people.freebsd.org/~dinoex/batch/pkg_jail
$ sh pkg_jail init

1.2. The source

To build kernel modules and more you need the ‘src’ tree mounted. This is done by default via ‘nullfs’ the ‘src’ tree from the host, or you can extract the ‘src’ inside the jail.

- ‘src’ tree needed.
  - use nullfs
  - or move ‘src’ from the base into the jail

1.3. Use the ports Luke

We move the ports tree inside our jail and create an symbolic link to it.

$ mv /usr/ports /usr/jail/build/usr/ports
$ ls -s /usr/jail/build/usr/ports /usr/

Instead you can also just create a new ‘ports’ tree inside the jail.

$ portsnap fetch
$ portsnap extract

1.4. Get Add-ons

We create an extra directory for our stuff.

```
$ mkdir -p /usr/ports/local/update
$ cd /usr/ports/local/update
$ fetch http://people.freebsd.org/~dinoex/batch/README
$ fetch http://people.freebsd.org/~dinoex/batch/pkg_update
```

1.5. Customizing.

We keep our ‘make.conf’ from the host.

$ cp /etc/make.conf /usr/jail/build/etc/make.conf

Usually we add to the file in both host and jail:

WRKDIRPREFIX?=/usr/tmp/obj
PACKAGES?=/usr/ports/packages-7-i386
USE_PACKAGE_DEPENDS=yes
DEPENDS_TARGET=package
BATCH=yes

Ignore files and directories when checking ‘plist’:

$ cd /usr/ports/local/update
cat >data/badfiles << ’EOF’
/usr/local/share/nls/POSIX
/usr/local/share/nls/en_US.US-ASCII
EOF
cat >data/baddirs << ’EOF’
/usr/local/share/nls
/usr/local/share
EOF

2. Ready to start

- 2.1. Get list of packages installed.
- 2.2. Collect the data for the jail.

2.1. Get list of packages installed.

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.

On the host we need to register what we have.

```
$ cd /usr/ports/local/update
$ sh pkg_update check-installed-ports
```

2.2. Collect the data for the jail.

We simply create the list of ports from the data we collected in the step before.

$ cd /usr/ports/local/update

$ cat data/install-packages.* >data/make-packages.build.local

3. Ready to build

We reuse packages until 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.

- 3.1. We build our first set of packages
- 3.2. Updating and makeing the new packages
- 3.3. Cleaning up
- 3.4. Fetch
- 3.5. Errors
- 3.6. Major updates

3.1. We build our first set of packages

Build ports and their dependencies in a clean /usr/local.

```
$ cd /usr/ports/local/update
$ pkg_update make-packages
```

3.2. Updating and makeing the new packages

Each dependency is checked for the exact version to allow updates and downgrades. 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 needed to track changes in the ‘build only’ dependencies.

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 ‘csup’ 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.

```
$ cd /usr/ports/local/update
$ pkg_update full-update-jail
```

3.2. Updating and makeing the new packages

Or run each step seperatly:

- Step 1: Start "make update" for the ports tree.
- Step 2: Find obsolete packages and move them away.
- Step 3: Rebuild the missing packages.

```
$ cd /usr/ports/local/update
$ pkg_update cvsup
$ pkg_update clean-packages
$ pkg_update make-packages
```

3.2. Updating and makeing the new packages

In case you urgent need a simple package with its dependencies, you can pass the name of the directory as an argument.

```
$ cd /usr/ports/local/update
$ pkg_update make-packages www/lynx sysutils/dmidecode
```

For often used subsets you can pass the name of a file which list the ports you need.

```
$ cd /usr/ports/local/update
$ pkg_update make-packages data/install-packages.test
```

3.3. Cleaning up

Once in a while you like to clean up your ‘distfiles’ directory. 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 to ‘/usr/ports/distfiles/Old/’.

```
$ cd /usr/ports/local/update
$ pkg_update clean-distfiles
```

3.4. Fetch

Usually the distfiles are fetched when needed. But you can script this easy before build to run chrooted with your external IP, while the build step runs with an IP of 127.0.0.1.

```
$ cd /usr/ports/local/update
$ pkg_update fetch-distfiles
$ pkg_update fetch-recursive-distfiles
```

3.5. Errors

In active development of the ports, logfiles are preserved in the log sub directory. 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
```

3.5. Errors

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.

```
-rw-r--r--  1 root  wheel    24890 Sep 15 18:13 build,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    57388 Sep 15 18:56 build,local,gworkspace
```

3.5. Errors

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.

```
-rw-r--r--  1 root  wheel      120 Sep 15 18:14 plist,local,gnumail
-rw-r--r--  1 root  wheel      161 Sep 15 18:38 plist,local,preferences.app
-rw-r--r--  1 root  wheel      198 Sep 15 18:57 plist,local,gworkspace
```

3.5. Errors

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.

```
-rw-r--r--  1 root  wheel      5290 Sep 16 06:25 err,local,gnumail_112
```

3.6. Major updates

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.

- Update your jail
  - installworld
  - mergemaster -i
  - make delete-old
  - make delete-old-libs

- Move all packages away

```
$ mv -i /usr/ports/packages7/All/* /usr/ports/packages7/Old/
```

- Then rebuildung all packages needed

```
$ cd /usr/ports/local/update
$ pkg_update make-packages
```

4. Updating the host

- 4.1. Preparation of the installed ports
- 4.2. Pre flight checks
- 4.3. Preparation for the update
- 4.4. Downtime

4.1. Preparation of the installed ports

In case we had meddled with ‘portupgrade’ on this host, we need to fix our installed ports registration.

```
$ cd /usr/ports/local/update
$ pkg_update dependency-update
```

After a while some ports gets moved or renamed. To ensure updates can continue, we need to fix the origin of the installed ports.

```
$ cd /usr/ports/local/update
$ pkg_update fix-moved-ports
```

4.2. Pre flight checks

See if our work list is still valid.

```
$ cd /usr/ports/local/update
$ pkg_update check-installed-ports
```

Test if we have all needed packages.

```
$ cd /usr/ports/local/update
$ pkg_update show-missing-packages
```

4.3. Preparation for the update

Now check if we are have old ports installed.

```
$ cd /usr/ports/local/update
$ pkg_update clean make-version-list
$ pkg_update show-version-list
```

4.3. Preparation for the update

In case of light updates we can try:

$ cd /usr/ports/local/update
$ pkg_update make-easy-update

We get a list of ordered ‘pkg_deinstall’ and ‘pkg_add’ commands to run in ‘easyupdate.${hostname}’. Check this file, it should be safe to run the commands, but usualy it does not reach deep to update all ports on your host.

4.3. Preparation for the update

If we have more work to do, this is the way to update hundreds of ports with its dependencies. We create a script to remove all obsolete ports and the ports that depends on them as ‘deinstall.${hostname}’.

$ cd /usr/ports/local/update
$ pkg_update make-deinstall-list

4.4. Downtime

Now we can stop some or all services on the host to do the real update. First the quick way:

```
$ cd /usr/ports/local/update
$ sh "easyupdate.${hostname}"
```

In this case we start over at 4.3.

4.4. Downtime

The complete update:

```
$ cd /usr/ports/local/update
$ sh "deinstall.${hostname}"
$ sh pkg_update reinstall
```

To check if we get everything you run the next command, usually the output is empty.

```
$ cd /usr/ports/local/update
$ sh pkg_update clean-reinstall
```

Then we can start all services again.

5. Extending your ports

- 5.1. Naming problems in ports
- 5.2. Local ports
- 5.3. Slave ports

5.1. Naming problems in ports

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’: This should be done on the host and in the jail, preferred by an include.

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

5.2. Local ports

To distinguish our local ports we create a Makefile.inc for them.

```
$ cat > /usr/ports/local/Makefile.inc << 'EOF'
PKGCATEGORY?=          local
PKGNAMESUFFIX?=        -local
PKGNAMESUFFIX2?=       -local
EOF
```

This done, we can copy any port into /usr/ports/local/ and modify is for your use.

5.3. Slave ports

Slave ports can be very easy created.

Example:
/usr/ports/local/mplayer-extra/Makefile:

```
MASTERDIR=      /usr/ports/multimedia/mplayer

WITH_SDL=yes
WITH_XANIM=yes
WITH_FREETYPE=yes

WITHOUT_RUNTIME_CPUDETECTION=yes
WITHOUT_3DNOW=yes

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

6. References

[1] http://www.dinoex.de/schulungen/package-en.html
[2] http://2004.eurobsdcon.org/uploads/media/EBSD04_27.pdf
[3] http://people.freebsd.org/~dinoex/batch/README
[4] http://people.freebsd.org/~dinoex/batch/pkg_update
[5] http://people.freebsd.org/~dinoex/batch/pkg_jail

7. Questions

Feedback: Dirk Meyer
[dirk.meyer@dinoex.sub.org],[dirk.meyer@guug.de],[dinoex@FreeBSD.org]