FreeBSD ports
(a personal perspective of a user)

Klaus Aehlig

May 10, 2011

Disclaimer

- It's a *personal* perspective
  and by no means an official position of the FreeBSD project.
  - All opinions expressed are my own.

- This is not a tutorial on the ports system.
  - This is just to give an idea, what it's about.
  - For full information read the Porter's Handbook.

- I'm using BSD only since October 2008,
  hence my experience is limited.
  - Probably, I'm not always doing things the best way.
  - What I tell can be inaccurate—or even wrong.

  BUT THERE'S MORE THAN ONE WAY TO DO IT!

The Problem

Every software project that involves more than a hand full of persons will inevitable have. . . *(note the order!)*

- persons with a difficult personality,
- bugs in parts important to you,
  but not important for most others,
- design choices that are not to your taste,
- . . .

There are no exceptions to this rule. FreeBSD isn't either.
(even though it's much better there than a lot other projects)

So, how to deal with that?

- Committees, rules, regulations, policies, standards, . . . ?
- Make it easy to deviate where you want *and only there!*

A Rant on Binary Distributions

I (personally) failed using an open-source binary distribution.
It was too much "One size fits all".

- Fixed policy on file system hierarchy layout, paths, ...
- Situations are different for various applications...
  - `strip` or `-g`?
  - Few library dependencies or full features?
  - Have X-support? Documentation?
  - ...

... but no global knobs like `WITHOUT_X11`, `WITH_DOC`, ...

- Changing a little thing meant forking the whole package.
- No easy way to adapt after switching a library version.

I don't want the system to tell me, what to do. I want it to adapt to my needs. So let's look at something different...

Ports—The Basic Idea

Essentially, a port is like a recipe. . .
or a formalised report of someone, who managed to install it

. . . you say what to do (buy ingredients, remove bad parts, . . .)

`fetch`, `checksum`, `extract`, `patch`, `configure`, `build`, `install`, `clean`

- All done with standard infrastructure: have a `Makefile`.
- Only write down what is specific to that very port!
  i.e., where you deviate from the vanilla `./configure && make && make install`; the rest is in a big shared file.
  `.include <bsd.port.mk>`
- That way 23k ports with 17y history fit into a single 1.7GB CVS repository.

But before we go into details, a little remark on `make`(1). . .

A detail on make(1)

> *First of all, the initial list of specifications will be read from the system makefile, sys.mk, unless inhibited with the* **-r** *option. The standard sys.mk as shipped with FreeBSD also handles make.conf(5), the default path to which can be altered via the make variable __MAKE_CONF.*
>
> *(man make)*

- Every call to make reads /etc/make.conf *(outside ‘pwd‘!)*
  . . . unless in an environment where you want something else.
- You can make the effect specific to a particular port using

```
.if !empty(.CURDIR:M*/ports/xxx/yyy*)
...
.endif
```

KEEP THIS IN MIND!

fetch, checksum, extract

Let's walk through misc/findutils. First: get the sources. Downloading is standard, so we only fill in the parameters.

```
PORTNAME= findutils
PORTVERSION= 4.5.9
MASTER_SITES= ${MASTER_SITE_GNU_ALPHA}
MASTER_SITE_SUBDIR= findutils
```

The files to fetch are DISTFILES, with default expanding to ${PORTNAME}-${PORTVERSION}${EXTRACT_SUFX}.
For obvious security reasons we store in distinfo

```
SHA256 (findutils-4.5.9.tar.gz) = ......
```

Files are fetched only once and stored in ${DISTDIR}. Check sums are checked. We unpack everything in ${WRKDIR}.

See all these variables? Remember we read /etc/make.conf?

Side remark: updating

- Note that the only thing we store that is particular to a version is the version number and the checksum
- So, for perfect upstream, updating is just
  - change the version number *(a single digit)*
  - `make makesum`
  - verify integrity of what you fetched
- In reality, before using (let alone showing anyone), you also want to
  - see how the build process has changed
  - verify how the set of installed files has changed
  - look for user-visible changes *(documented and undocumented)*
  - check for bugs *(and communicate fixes back upstream)*
  - ...

patch, configure

- patches from ${PATCHDIR} are applied
- configure is run *(this is also a good place to honour options)*

```
GNU_CONFIGURE=yes
CONFIGURE_ENV= CPPFLAGS="-I${LOCALBASE}/include" ...
.if !defined(WITHOUT_NLS)
USE_GETTEXT= yes
PLIST_SUB+= NLS=""
.else
CONFIGURE_ARGS+= --disable-nls
PLIST_SUB+= NLS="@comment "
.endif
CONFIGURE_ARGS+= --program-prefix=g ...
```

Note: the list of files installed changes depending on options

build, install, clean

- the build utility is called to build; usually also for install
  but sometimes need a `do-install` target

  ```
  USE_GMAKE= yes
  MAKE_ARGS= INSTALL_SCRIPT="${INSTALL_SCRIPT}"
  MAKE_JOBS_SAFE= yes
  ```

- After installation, the software is registered with its file list
  (essentially the file `pkg-plist`, with `PLIST_SUB` honoured; but also consider `INFO`, `MAN1`, ..., `PLIST_FILES`, ...)
  - Hashes of all installed files are computed.
  - install/remove scripts also in `pkg-plist`
  - Also: `pkg-descr`, `COMMENT`, ...
  - actual dependencies are registered
  - ...

  Home-grown ports trees may shortcut here, if `stow(1)` is used as packaging tool.
- `clean` is easy. Just throw away `${WRKDIR}`

Dependencies

- Distinguish between `FETCH_DEPENDS`, `EXTRACT_DEPENDS`, `PATCH_DEPENDS`, `BUILD_DEPENDS`, `RUN_DEPENDS`, `LIB_DEPENDS`.
- given as a triple
  - A file that must exists (maybe in `${PATH}`), a library (maybe with version constraints), ...
    Note: dependency can be provided by a different than the intended package
  - a port directory for the dependency
  - a target to execute, in order to get the dependency
    usually omitted, if the default `install` applies

And, of course, there is `NO_DEPENDS` for the user to override...

Slave Ports

Remember? It's all about setting variables right. . .
So with ?= in the right places, you can be useful for someone else.

The whole(!!) port `print/a2ps-a4` reads as follows.

```
PAPERSIZE= a4
MASTERDIR= ${.CURDIR}/../a2ps-letter

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

Again: only describe what's different.

EXTRA_PATCHES

- That `${PATCHDIR}/patch-*` is applied is only half the truth
  . . . as this would be much to inflexible!
- There are “distribution patches” (provided by 3rd party).
  Don’t duplicate code!
  Set PATCH_FILES and PATCH_SITES for that.
- Some patches are only for certain user options.
  Some distribution patches need preprocessing. . . .

⇝ Can set `EXTRA_PATCHES` for that.

⇝ And then there are the targets `pre-patch`, `post-patch`, . . .

- But there are also creative uses of all these. . .

EXTRA_PATCHES (mis)used for site-patches

Say, on your machine, you want a different greeting for `gunits(1)`.

- `cd /usr/ports/math/units && make extract`
- copy `units.c` to `units.c.orig` and change `units.c`
- `diff -u units.c.orig units.c > /x/y/z.diff`
- add to `make.conf`

```
.if !empty(.CURDIR:M*/ports/math/units*)
EXTRA_PATCHES += /x/y/z.diff
.endif
```

- Reinstall as usual (`portupgrade -f units`) and...

```
$ gunits
This program contains a patch by Klaus
2526 units, 72 prefixes, 56 nonlinear units
...
```

- Note: nothing changed under `/usr/ports`!
  So, you get updates as usual, with your usual update-routine.

Flexibility. . .

. . . sometimes requires a bit of extra work.

```
post-patch:
@${REINPLACE_CMD} -e "s|/usr/local|${PREFIX}|" \
${WRKSRC}/examples/config/config \
${WRKSRC}/bin/uzbl-browser \
${WRKSRC}/bin/uzbl-event-manager
@${REINPLACE_CMD} -e
"s|share/uzbl|${DATADIR_REL}|" \
${WRKSRC}/examples/config/config \
${WRKSRC}/bin/uzbl-browser \
${WRKSRC}/bin/uzbl-event-manager
@${REINPLACE_CMD} -e
"s|/usr/share/uzbl|${DATADIR}|" \
${WRKSRC}/bin/uzbl-tabbed
```

But it's worth the extra effort!

A word to everyone distribution free open-source software

> *NOTES*
> *This manual page documents the default FreeBSD file system layout, but the actual hierarchy on a given system is defined at the system administrator's discretion.*
>
> *(man hier)*

People do will change things according to their needs.
That's the whole point of open-source!

$\rightsquigarrow$ By relying on a fixed layout/policy/. . . you're working *against* your users, as you make it hard for them to get their job done *(which might be different from your goals).*

Finally. . .

. . . it's the ideas that matters, not the concrete implementation!

- Don't duplicate, only document where you deviate
  . . . and why you had to.
- Respect the local system administrator.
  With one computer per person here, that is: the end user.

⇝ Honour `PREFIX`, `LOCALBASE`, . . . TOOLS, NOT POLICIES.

I've got my own little ports tree for my GNU/Linux machines.

- On our server, we sometimes can't use the distribution.
  - Packet too far away from upstream.
  - We need a specific version.
  - We need patches very specific to our machine.
  - . . .
- It was also useful, when I had to use a machine, where I disagreed with the administrator ;-)