Hot cross builds
Cross-compilation in pkgsrc

Taylor R Campbell
riastradh@NetBSD.org

BSDCan 2024
Ottawa, Canada
May 31, 2024

Hot cross builds: cross-compilation in pkgsrc

https://www.NetBSD.org/gallery/presentations/riastradh/bsdcan2024/pkgcross.pdf

pkgsrc: portable package build system

- `https://pkgsrc.org/`
- Framework for building third-party software on Unix-like operating systems.
- >26,000 packages.
- Actively supported platforms:
  - NetBSD (first platform, based on mid-'90s FreeBSD ports)
  - Solaris/SmartOS/illumos
  - Linux
  - macOS
- Other platforms with some support:
  - FreeBSD/OpenBSD/DragonflyBSD/MidnightBSD
  - MINIX 3
  - SCO OpenServer/UnixWare
  - HP-UX
  - QNX
- Works unprivileged, so you can develop in your home directory on a server you don't administer.

Anatomy of a pkgsrc package

- `DESCR` – Human-readable description.
- `Makefile` – Machine-readable description.
  - Tells where to download source code.
  - Rules for how to configure, build, install.
  - Etc.
- `distinfo` – Names, sizes, and hashes of source distribution. Provides cryptographic integrity check.
- `PLIST` – Packing list: lists files installed by package.

pkgsrc example: security/nettle, part 1

```
# $NetBSD: Makefile,v 1.31 2023/06/06 05:12:06 adam Exp $

DISTNAME=       nettle-3.9.1
CATEGORIES=     security
MASTER_SITES=   http://www.lysator.liu.se/~nisse/archive/
MASTER_SITES+=  ftp://ftp.lysator.liu.se/pub/security/lsh/

MAINTAINER=     pkgsrc-users@NetBSD.org
HOMEPAGE=       https://www.lysator.liu.se/~nisse/nettle/
COMMENT=        Cryptographic library
LICENSE=        gnu-lgpl-v2.1

USE_LANGUAGES=          c c99
USE_LIBTOOL=            yes
USE_TOOLS+=             gm4 gmake
GNU_CONFIGURE=          yes
SET_LIBDIR=             yes
CONFIGURE_ARGS+=        --disable-openssl
CONFIGURE_ARGS+=        --disable-shared
```

pkgsrc example: `security/nettle`, part 2

```
.include "../../mk/bsd.prefs.mk"

.if ${USE_CROSS_COMPILE:tl} == "yes"
CONFIGURE_ENV+=         CC_FOR_BUILD=${NATIVE_CC:Q}
.endif

INFO_FILES=             yes
TEST_TARGET=            check
PKGCONFIG_OVERRIDE=     hogweed.pc.in
PKGCONFIG_OVERRIDE+=    nettle.pc.in

BUILDLINK_API_DEPENDS.gmp+=     gmp>=6.0
.include "../../devel/gmp/buildlink3.mk"
.include "../../mk/bsd.pkg.mk"
```

Building and installing a package[1]

```
# which socat
socat not found
# cd /usr/pkgsrc/net/socat
# bmake install
=> Bootstrap dependency digest>=20211023: found digest-20220214
=> Fetching socat-1.8.0.0.tar.gz
...
=> Checksum SHA512 OK for socat-1.8.0.0.tar.gz
===> Installing dependencies for socat-1.8.0.0
...
=> Tool dependency checkperms>=1.1: found checkperms-1.12
=> Full dependency readline>=6.0: found readline-8.2nb2
...
=> Creating binary package .../socat-1.8.0.0.tgz
===> Installing binary package of socat-1.8.0.0
# which socat
/usr/pkg/bin/socat
```

[1] On NetBSD, can use base system's `make`, but everywhere else we bootstrap `devel/bmake` for pkgsrc.

Binary packages: build once, install many times

- Building from source is necessary: verify source, audit programs, modify, etc.
- Building from source is slow: run compiler on lots of source code.
- Do it once, save the result, install binary packages after.

```
builder$ cd /home/builder/pkgsrc/net/socat
builder$ bmake package

client# PKG_PATH=/nfs/builder/pkgsrc/packages
client# export PKG_PATH
client# pkg_add socat
client# which socat
/usr/pkg/bin/socat
```

Binary package bulk builds

- NetBSD provides binary packages for NetBSD on many architectures[2].
- MNX Cloud provides binary packages for SmartOS, macOS, Linux, and NetBSD/amd64[3].
- I build binary packages for my own machines.
- You can too!

[2] https://ftp.NetBSD.org/pub/pkgsrc/packages/NetBSD/

[3] https://pkgsrc.smartos.org/

Cross-compiling NetBSD

- Every NetBSD build is a cross-build.
- `build.sh tools` builds cross-toolchain.
- `build.sh kernel=GENERIC distribution` builds NetBSD with the cross-toolchain.

Cross-compiling pkgsrc

- Use NetBSD `build.sh tools distribution` to get started.[4]
- `USE_CROSS_COMPILE=yes`
- `TOOLDIR=/usr/obj.evbppc/tooldir.NetBSD-10.0-amd64`
- `CROSS_DESTDIR=/usr/obj.evbppc/destdir.evbppc`
- `CROSS_MACHINE_ARCH=powerpc`, `CROSS_OPSYS=NetBSD`, ...

```
$ uname -m
amd64
$ cd ~/pkgsrc/net/socat
$ bmake package
...
$ cd ~/pkgsrc/packages.NetBSD-10.0-powerpc/All
$ pkg_info -Q MACHINE_ARCH socat-1.8.0.0.tgz
powerpc
```

[4] See `doc/HOWTO-use-crosscompile` for details.

Cross-build in homedir, install systemwide on target

- `./bootstrap --prefix /home/builder/pkg --unprivileged ...`
- set `CROSS_LOCALBASE=/usr/pkg` in mk.conf

Toolchain wrappers

- pkgsrc creates symlink farms of toolchain wrappers for build:
  - `cc`, `ld`, `as`, ...
  - `powerpc--netbsd-gcc`, `powerpc--netbsd-ld`, `powerpc--netbsd-as`, ...
- pkgsrc buildlink3 framework creates symlink farms of dependent header files and libraries for build isolation.
- Wrappers transform toolchain arguments:
  - add `--sysroot=${CROSS_DESTDIR}`
  - ensure `-I` (build-time include path) and `-L` (build-time library path) point at buildlink3 symlink farms
  - ensure `-Wl,-R` (run-time library path) points at installation prefix without `CROSS_DESTDIR`
  - replace `-ldl` by appropriate platform-specific dlfcn.h option
  - other package-specific argument transformations

Dependencies

- Some packages **depend** on other packages:
  - `tor` program needs `libevent` library at run-time
    - `net/tor` (**run-**) **depends** on `devel/libevent`
  - Compiler needs `event.h` when building `tor` program at compile-time
    - `net/tor` also **build-depends** on `devel/libevent`
  - Building `libxcb` requires running `xsltproc` to turn XML into C header files at compile-time
    - `x11/libxcb` **tool-depends** on `textproc/xsltproc`
  - Also **bootstrap-depends**, like tool-depends but for parts of the pkgsrc infrastructure.

Cross-compiling dependencies

- Use Intel Xeon to build `x11/xterm`, run on your powerpc-based thin client.
- `x11/xterm` must be *cross-built* for `MACHINE_ARCH=powerpc`.
- `x11/xterm` depends on `x11/libxcb`[5].
  - `x11/libxcb` must be *cross-built* for `MACHINE_ARCH=powerpc`.
- `x11/libxcb` *tool-depends* on `textproc/xsltproc`.
  - `textproc/libxsltproc` must be *natively built* for `MACHINE_ARCH=x86_64`.

[5] Via `x11/libX11`.

Build-depends vs tool-depends

- Both build-depends and tool-depends need to exist at build-time.
- *Build-depends* are cross-built and installed into `/usr/obj.evbppc/destdir.evbppc/usr/pkg/...`
  - Example: C libraries, needed for linker.
- *Tool-depends* are natively built and installed into `/home/builder/pkg/...` (`${TOOLBASE}`)
  - Example: `xsltproc`, cross-compiler.
  - `TARGET_MACHINE_ARCH`, `TARGET_OPSYS`, ..., are set to cross-compilation target.

Pointing builds at tool programs in dependencies

- Package uses glib-mkenums at build-time, how to use it?

  ```
  TOOL_DEPENDS+=                \
      glib2-tools>=0:../../devel/glib2-tools
  ```
- GNU Autoconf:

  ```
  CONFIGURE_ARGS+=              \
      GLIB_MKENUMS=${TOOLBASE:Q}/bin/glib-mkenums
  ```
- Meson:

  ```
  MESON_CROSS_BINARIES+=  glib-mkenums
  MESON_CROSS_BINARY.glib-mkenums=        \
      ${TOOLBASE}/bin/glib-mkenums
  ```
- Similarly: Use `TOOL_PYTHONBIN` at build-time, but bake `PYTHONBIN` into product for run-time Python.

Meson: pkgsrc creates cross config for you

```
[properties]
sys_root = '/usr/obj.evbppc/destdir.evbppc'
[host_machine]
system = 'netbsd'
cpu_family = 'ppc'
cpu = 'powerpc'
endian = 'big'
[binaries]
glib-genmarshal = '/home/builder/pkg/bin/glib-genmarshal'
glib-mkenums = '/home/builder/pkg/bin/glib-mkenums'
```

Complications part 1: mixing up build-depends and tool-depends

- Originally, pkgsrc had only build-depends—same as tool-depends for native builds.
  - `x11/libxcb` build-depended on `textproc/xsltproc`.
- Packages practically never need to set `BUILD_DEPENDS` directly—only via buildlink3.
- Solution: We mass-changed `BUILD_DEPENDS` to `TOOL_DEPENDS` in package makefiles.

Complications part 2: package builds tools internally

- Some packages depend on external tools like `x11/libxcb` depends on `textproc/xsltproc`.
- Others use internal tools, like `security/nettle` above.
- These try to use `CC`, which may be `powerpc--netbsd-gcc` for cross-compilation.
- Can't run the result on x86!
- Solution: Set `CC_FOR_BUILD`, maybe patch package to use it instead.

```
.include "../../mk/bsd.prefs.mk"

.if ${USE_CROSS_COMPILE:tl} == "yes"
CONFIGURE_ENV+= CC_FOR_BUILD=${NATIVE_CC:Q}
.endif
```

Complications part 2′: package runs its own build product

- Some packages want to run a program they also install.
  - `x11/gtk2` calls `gtk2-update-icon-cache`.
- Need both native *and* cross versions of the program!
- Solution: Have package tool-depend on itself and pass path to the natively built tool in the cross-build:

```
.include "../../mk/bsd.prefs.mk"

.if ${USE_CROSS_COMPILE:tl} == "yes"
TOOL_DEPENDS+=      ${PKGNAME}:../../${PKGPATH}
UPDATE_ICON_CACHE=  \
    ${TOOLBASE:Q}/bin/gtk2-update-icon-cache
CONFIGURE_ENV+=     \
    GTK2_UPDATE_ICON_CACHE=${UPDATE_ICON_CACHE}
.endif
```

Complications part 3: file existence tests

- Package wants to know whether `/dev/urandom` will exist when run.
- Uses GNU Autoconf to ask whether `/dev/urandom` exists *now*, when built.
- Build machine and target system may be different!
- But we know `/dev/urandom` will exist.
- Solution: Tell configure up front:

```
.include "../../mk/bsd.prefs.mk"

.if ${USE_CROSS_COMPILE:tl} == "yes"
CONFIGURE_ENV.NetBSD+=  ac_cv_file__dev_urandom=yes
.endif
```

Complications part 3′: file existence tests in pkgsrc

- From `x11/libdrm`:

```
.if !exists(/usr/include/sys/atomic.h)
# libdrm won't find system atomic ops, use a package.
.  include "../../devel/libatomic_ops/buildlink3.mk"
.endif
```

- Solution: Don't look in `/usr/include` — look in `/usr/obj.evbppc/destdir.evbppc`:

```
.include "../../mk/bsd.prefs.mk"

.if !exists(${_CROSS_DESTDIR}/usr/include/sys/atomic.h)
# libdrm won't find system atomic ops, use a package.
.  include "../../devel/libatomic_ops/buildlink3.mk"
.endif
```

Complications part 4a: configure run-tests

- Similar to file existence tests.
- Program wants to know `sizeof(long)` at compile-time.
- Compiles a test program to print it, runs test program.
- Can't do that if building on 64-bit amd64 for 32-bit powerpc!
- Solution: Binary search with compile-time assertions using cross-compiler.
- (Yes, seriously! GNU Autoconf supports this with `AC_CHECK_SIZEOF`.)

Complications part 4b: configure run-tests

- Some are harder to replace.
- Tell the answers up front, maybe with patches.
- From `shells/zsh`:

```
.include "../../mk/bsd.prefs.mk"

.if ${USE_CROSS_COMPILE:tl} == "yes"
.if ${OPSYS} == "NetBSD"
CONFIGURE_ENV+= zsh_cv_shared_environ=yes
CONFIGURE_ENV+= zsh_cv_shared_tgetent=yes
CONFIGURE_ENV+= zsh_cv_shared_tigetstr=yes
CONFIGURE_ENV+= zsh_cv_sys_dynamic_execsyms=yes
.endif
.endif
```

Complications part 5: problem children

- Some packages go to great effort to resist cross-compilation.
  - Perl
  - Python
  - gobject-introspection
- Workaround: just build on your powerpc thin client and ship binary packages back to x86 build machine to continue.
- (Solution: Chainsaws and rototillers. Fix the build systems![6])

[6]It can be done for Perl: OpenWrt does it. If you would like to help adapt their approach to pkgsrc, talk to me!

Complications part 5: problem children

- Some packages go to great effort to resist cross-compilation.
  - Perl
  - ~~Python~~
  - gobject-introspection
- Workaround: just build on your powerpc thin client and ship binary packages back to x86 build machine to continue.
- (Solution: Chainsaws and rototillers. Fix the build systems![6])

[6] It can be done for Perl: OpenWrt does it. If you would like to help adapt their approach to pkgsrc, talk to me!

Complications part 5: problem children

- Some packages go to great effort to resist cross-compilation.
  - Perl
  - ~~Python~~ (much better since 3.10)
  - gobject-introspection
- Workaround: just build on your powerpc thin client and ship binary packages back to x86 build machine to continue.
- (Solution: Chainsaws and rototillers. Fix the build systems![6])

[6] It can be done for Perl: OpenWrt does it. If you would like to help adapt their approach to pkgsrc, talk to me!

Related work

- OpenWrt: cross-compiled packages for Linux-based network appliances.
  - Linux-only.
  - Not general-purpose package system.
  - Much smaller than pkgsrc.
- `distcc`: run pkgsrc on thin client, run compiler remotely on x86 build machine.
  - Complex to set up: many moving parts (literally).
  - Hard to parallelize.
  - Compiler is a big part but not all of run-time—make(1) is a big part of pkgsrc cost.
- FreeBSD ports: run native compiler in user-mode emulator.
  - Many moving parts (figuratively).
  - Emulators are slow.
  - Less clean separation between host and target.

Future work

~~Future~~ Past work

(since AsiaBSDcon 2015)

~~Future~~ Past work

- Cross-OS compilation. Use SmartOS x86 cloud cluster to build for `MACHINE_PLATFORM=NetBSD-7.0-powerpc`.

~~Future~~ Past work

- Cross-OS compilation. Use SmartOS x86 cloud cluster to build for `MACHINE_PLATFORM=NetBSD-7.0-powerpc`.
  - Set both `CROSS_MACHINE_ARCH` and `CROSS_OPSYS` in mk.conf.

~~Future~~ Past work

- Cross-OS compilation. Use SmartOS x86 cloud cluster to build for `MACHINE_PLATFORM=NetBSD-7.0-powerpc`.
  - Set both `CROSS_MACHINE_ARCH` and `CROSS_OPSYS` in mk.conf.
  - Still to fix: `USE_TOOLS+= ...:run`. pkgsrc doesn't distinguish host OS from target OS in `USE_TOOLS`.

~~Future~~ Past work

- User interface improvements.

~~Future~~ Past work

- User interface improvements.
    - Can't do `bmake package MACHINE_ARCH=powerpc` for stupid reasons.
    - Setting up cross-compiling requires a manual step to work around broken GNU `libtool`.

~~Future~~ Past work

- User interface improvements.
  - ~~Can't do `bmake package MACHINE_ARCH=powerpc` for stupid reasons.~~
    - `bmake package CROSS_MACHINE_ARCH=powerpc`
  - ~~Setting up cross-compiling requires a manual step to work around broken GNU `libtool`.~~
    - Bug fixed!

~~Future~~ Past work

- Bulk builds.
  - `pbulk` doesn't understand build-depends vs tool-depends.

~~Future~~ ~~Past~~ Future work

- Bulk builds.
  - `pbulk` doesn't understand build-depends vs tool-depends.

~~Future~~ Past work

- Unprivileged builds for privileged installs.
  - Native and cross packages must both point at `/usr/pkg`.
  - (Unprivileged builds for unprivileged installs work fine—not a problem with privileges, just with different paths.)

~~Future~~ Past work

- Unprivileged builds for privileged installs.
  - ~~Native and cross packages must both point at `/usr/pkg`.~~
    - `LOCALBASE=/home/builder/pkg` and `CROSS_LOCALBASE=/usr/pkg` in the same mk.conf.
  - (Unprivileged builds for unprivileged installs work fine—not a problem with privileges, just with different paths.)

~~Future~~ Past work

- Unprivileged builds for privileged installs.
  - ~~Native and cross packages must both point at /usr/pkg.~~
    - `LOCALBASE=/home/builder/pkg` and `CROSS_LOCALBASE=/usr/pkg` in the same mk.conf.
  - (Unprivileged builds for unprivileged installs work fine—not a problem with privileges, just with different paths.)
  - Some remaining issues: `chown` tool, suid executables.

Now get cross-building!

Questions?