Why and Goals

Overview

- Overview of the build
- Improvements to the build
- Planned improvements
- Wishlist

OVERVIEW OF THE BUILD

Building recursively

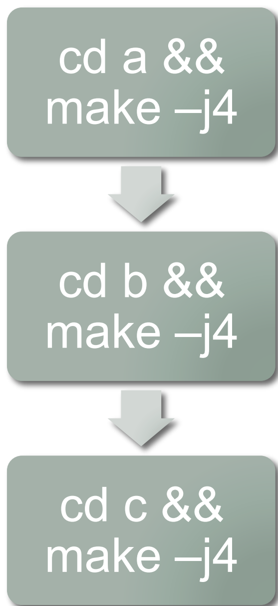
Recursive make considered harmful

- Build targets split into multiple directories
  - bin/ sbin/ lib/ usr.bin/ etc/
- Visited by *tree walks* using recursive **make all**
- Order determined by **SUBDIR** lists
  - bin/Makefile: SUBDIR= sh cat chflags …
- Generally one directory per **SUBDIR** list is built at a time unless **SUBDIR_PARALLEL** is defined (10.0+)
  - lib/Makefile: **SUBDIR_DEPEND_libc= libcompiler_rt**
- Subdirectory builds will not build dependencies from other directories
- Top-level has extra hacks to build cross-directory dependency graphs using subdir-build __*L* targets
  - gnu/lib/libgcc__L: lib/libc__L

Building recursively in parallel

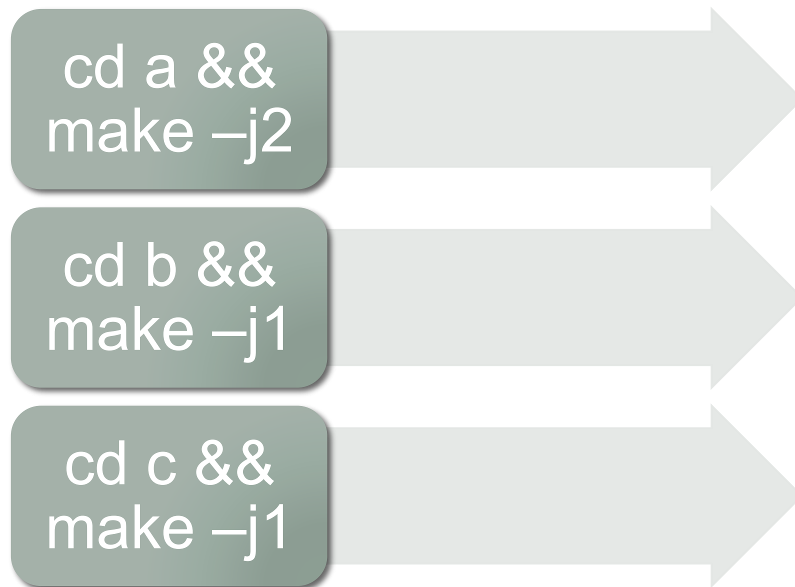
Serial build (–j4)

SUBDIR= a b c



SUBDIR_PARALLEL (–j4)

SUBDIR= a b c



**(Job handling is not accurate)*

Building FreeBSD

- **make buildworld**
  - Used for building the FreeBSD userland
  - The only reliable way to build
  - Bootstraps everything needed for the build
  - Overall a simple recursive build but very complex in the details
- **make buildkernel**
  - A non-recursive build using a massive Makefile generated by **config** that reads *sys/conf/files*
  - Modules are built recursively
- **make universe**
  - Builds **buildworld** and **buildkernel** for all supported architectures and kernel configs

Buildworld

- Has a “minimum supported release” (*MSR)*
  - Only supports building from FreeBSD
  - Documented upgrade policy is only from last Major Release
  - Buildworld on head (11.0) supports building from ~9.1 currently
- Supports *cross-build* to build binaries for a different architecture than the host
  - Must build native host binaries for build tools
  - Typically done even for native builds for reproducibility
  - *Host build*
    - Uses host headers and libraries
  - *Target* build
    - Uses a *sysroot* of staged headers and libraries from the tree
    - Builds using a *Toolchain* tailored for the target

Buildworld

Bootstrapping ad nauseam

- **make legacy** (Host)
  - Libegacy for compatibility shims to meet *MSR* for build tools.
- **make bootstrap-tools** (Host)
  - Updated version of tools to support *MSR*
- **make clean**
  - If not using **–DNO_CLEAN**
  - Avoids broken incremental build (csu, tools, CFLAGS, …)
- **make obj**
  - Creates a temporary object directory for everything to build into
- **make build-tools** (Host)
  - Some directories have special tools/generators needed to build themselves that are built here
- **make cross-tools** (Host)
  - Builds the *Toolchain* with default *Target/sysroot* set to $WORLDTMP ($OBJDIR/tmp)

Continued…

Buildworld

Bootstrapping ad nauseam

- **make includes (Target)**
  - Stages all headers into $WORLDTMP
- **make libraries (Target)**
  - Builds all libraries for later linkage and stages into $WORLDTMP
  - Multi-phase bootstrappy itself: *_prereq_libs, _startup_libs, _prebuild_libs, _generic_libs*
  - Uses *directory/libname__L:* targets to build dependency graph and does a non-recursive build of some, and a recursive of lib/ directories.
- **make depend (Target)**
  - Runs the preprocessor to build *.depend* files for clean parallel builds and for later incremental build
    - .depend: foo.o: /usr/include/stdlib.h /usr/include/stdio.h …
- **make everything (Target)**
  - Finally builds everything via **make all**, including libraries and a compiler “again”.
- **make libcompat (Target)**
  - Lib32 (64bit targets) and Libsoft (armv6)

Installing

- **make installworld**
  - Installs userland
- **make installkernel**
  - Installs the kernel and modules

Improvements to the Build

WITH_DIRDEPS_BUILD

- Presented by Simon Gerraty in 2014, merged BSDCan 2015
- Originally named *WITH_META_MODE* but renamed so that name could be used for something else
- A non-recursive build that works and builds dependencies from top-level or a subdirectory
- Uses checked-in *Makefile.depend* files in every directory that are auto generated from *filemon(4)* data containing a list of directories to build before the current one
- Avoids all of the *tree walks* and dependency hell lists from **buildworld**
- Optional dependencies are a problem, so not as viable for a default build
- Very useful for downstream vendors who have a static option list
- Its foundation brought in several features that are useful for **buildworld**, etc.

Parallel install

- **make installkernel –j**
- **make installworld –j**
  - Historically not safe to do but now *mostly* safe.
  - Installs rtld, and then libc first before installing the rest of the system.
  - A proper install would be a full dependency-ordered install
  - The install order is actually more correct in parallel due to *SUBDIR_DEPEND* lines vs *serially* as the *SUBDIR* ordering is unintentionally no longer correct after the addition of *SUBDIR_PARALLEL* to *lib/Makefile*
    - Fixing this could be done through making *serial* traversals in bsd.subdir.mk respect *SUBDIR_DEPEND* lines.

STANDALONE_TARGETS

Improving simple tree walks

- Some targets done in *tree walks* will not have any interdependencies
- Always build in parallel (SUBDIR_PARALLEL) without using any *SUBDIR_DEPEND*
- **make** all-man buildconfig buildfiles buildincludes check checkdpadd **clean** cleandepend cleandir cleanilinks cleanobj files **includes** installconfig installincludes installfiles maninstall manlint **obj** objlink
- When building with **–DNO_ROOT** (images) then **make install** is also ran in parallel
- Defined in *share/mk/bsd.subdir.mk*
- Can be appended to in src.conf for downstream

WITH_CCACHE_BUILD

Compiler output caching

- Uses **ccache** for all compliations
- Why built-in and not *PATH* or *CC*?
  - No preprocessor cache
  - No assembler cache
  - No linking cache
- Only helps for broken or overly aggressive incremental builds
- Stats for a clean **make buildworld** (95% confidence, ZFS):
  - 20% slower on empty cache
  - 51% faster with full cache
    - Reveals a lot of overhead such as **make depend**, *tree walks* and *sysroot* staging
  - 66% faster with full cache and *WITH_FAST_DEPEND*

WITH_FAST_DEPEND

How it all worked before

- Normally **make depend** runs **cc –E** and stores the contents in *.depend*
  - Target.o: /usr/include/stdlib.h /usr/include/stdio.h …
  - Preprocessor ran again during compilation in **make all**; the preprocessed *.i* files are not stored or reused
- **make depend** also ensures all files are generated for build
- *.depend* files are used to apply header dependencies to proper source files
  - Without a *.depend* then all source files are assumed to depend on all headers: **${OBJS}: ${SRCS:M*.h}** to allow clean parallel builds
- **DPSRCS** used to contain generated files
- *.depend* also contains static prog dependencies:
  - cat.full: /usr/lib/libc.a

WITH_FAST_DEPEND

How it works now

- No more **make depend** *tree walk*
- *.depend.target.o* files generated during compilation with GCC 3.0-era –*MM* flags.
- The *.depend.target.o* files are now only useful for incremental builds
- Clean parallel builds rely on **beforebuild** hook on **make depend** to generate all source files before building any objects
- A *.depend* is still generated for static prog dependencies
- Stats (95% confidence, ZFS):
  - **make buildworld**: 16% time saved
  - **make buildkernel**: 35% time saved

WITH_FAST_DEPEND

Details

- The –*MM* flags are only applied to objects that are in OBJS/POBJS/SOBJS/DEPENDOBJS from DEPENDSRCS/SRCS
  - Avoids generating dependency files for special cases like the csu build that don't need them or introduce duplicate depencencies that confuse SUFFIX rules and result in multiple source files being compiled at once
- *OBJS_DEPEND_GUESS* and *OBJS_DEPEND_GUESS.target.o* can be used to add a dependency to an object target if there is no *.depend.target.o* for it yet
- Files included by new bmake feature **.dinclude<>** done from *bsd.dep.mk* directly rather than generating a loop inside of *.depend*
  - *.depend* inclusion in make has special handling for dependencies on missing files
    - *"Ignoring stale dependency"*

WITH_FAST_DEPEND

Downstream changes needed

- *DPSRCS* is not really needed anymore
  - Mostly just headers in it but they can safely be in *SRCS* (for many years now)
  - Special dependencies (generators) should not be in *SRCS* or *DPSRCS*, just create actual dependency rules for them
    - file.c: generator
      - ./generator > ${.TARGET}
    - generator: generator.c
      - ${CC} …
- The removal of **make depend** *tree walk* can harm some downstream builds that rely on a 2-pass parsing of Makefiles but should be very rare

WITH_SYSTEM_COMPILER

Opportunistically building clang less

- **make buildworld** normally builds clang twice: bootstrap (Host) and the one to be installed **(Target)**
- **make kernel-toolchain** normally builds clang once: bootstrap (Host)
- **make universe** normally builds clang N*2 times, where N is the number of architectures supported, for the same reason as **buildworld**. N of those are the bootstrap version with the only difference being the default *sysroot* and *target* architecture.
- Why build a bootstrap one at all rather than use **/usr/bin/cc**?
  - Major version upgrades
  - Bug fixes
  - Newly supported CFLAGS (like -fformat-extensions)
  - Reproducibility

WITH_SYSTEM_COMPILER

Opportunistically building clang less

- *__FreeBSD_cc_version* is incremented on any change to the compiler that warrants rebuilds, and for adding new architecture **–target** support.
- If the major version and *__FreeBSD_cc_version* of **${CC}** matches what is stored in the tree, then just use it as an *external compiler*.
  - This adds in ***-target*** and ***--sysroot*** flags into the build to build for the given *TARGET.TARGET_ARCH* and the build's *sysroot*.
  - GCC does not support **–target** so this logic is only used if building for the same architecture as the host. A bootstrap cross-compiler for cross architecture builds is still needed.
- Not the same as *WITHOUT_CROSS_COMPILER*, which _always_ builds with /usr/bin/cc.

WITH_SYSTEM_COMPILER

Getting the versions

- In tree
  - *__FreeBSD_cc_version* fetched from tree with **awk**
    - *lib/clang/freebsd_cc_version.h*
      - *#define FREEBSD_CC_VERSION*
    - *gnu/usr.bin/cc/cc_tools/freebsd-native.h*
      - *#define FBSD_CC_VER*
  - Major version fetched from tree with **awk**
    - *lib/clang/include/clang/Basic/Version.inc*
      - *#define CLANG_VERSION*
    - *contrib/gcc/BASE-VER*
- **${CC}**
  - *echo "__FreeBSD_cc_version" | ${CC} -E - | tail –n 1*

Automatic object directory creation

WITH_AUTO_OBJ

- Create object directory without needing **make obj** first
  - Avoids an expensive *tree walk* for **make buildworld**
  - Avoids mistakes of building without an object directory and having files in both the source directory and object directory. That can break **buildworld** as well.
- Works for the *WITH_DIRDEPS_BUILD* build system already as it was imported for that feature
- Not yet ready for subdirectories / **make buildworld** but close
- Also with this change comes changing OBJDIR to be
  - **${MAKEOBJDIRPREFIX}/${SRCTOP}/${TARGET}.${TARGET_ARCH}/${RELDIR}**
    - **/usr/obj/usr/src/amd64.amd64/bin/sh**
  - This organizes the OBJDIR for multiple checkouts better

Filemon(4)

Track all the dependencies

- Originally implemented by the late John Birrell and Juniper in 2009
- A ton of performance improvements and stability fixes have gone into it recently
- Tracks all files read/written/executed during the execution of a process
- It does what GCC –*MM* does but for everything
- Available from a C API, **script(1)**, and bmake
- Creates a log
  - E /bin/sh
  - R /usr/include/stdio.h
  - W /usr/obj/usr/src/bin/sh/sh.full
- **script –f log command**
  - log.filemon

Filemon(4)

Changes and remaining work

- Fixed bugs
  - Looping on all processes in syscalls looking for filemon tracer, now uses *struct proc.p_filemon*
    - This makes the module no longer self-contained but is worth it for performance
  - Many races
  - Error handling
  - Credential handling
- Todo
  - Stop using syscall hooks by improving *EVENTHANDLER(9)* or adding a new syscall trace framework
  - Will allow unloading
  - Some at(2) functions are missing or improperly handled

Bmake Meta Mode

Rebuild cases

- Presented by Simon Gerraty in 2014
- *.MAKE.MODE=meta*
- Provides functionality to have a reliable incremental build without cleaning
- Creates a *target.o.meta* file for every target as it is built
- Considering its *.meta* file, rebuilds a target if:
  - The command to build changes from last time
    - Such as different *CFLAGS* or a different path'd compiler
  - Files read, written, executed or linked are missing
    - Written is also important for staging
  - Filemon data is not present and filemon is enabled
  - A *.meta* file is missing (enabled the feature vs last build not enabled)
  - Files read/executed/linked to are newer than the target

Bmake Meta Mode

Meta file example

```
# Meta data file /usr/obj/root/git/freebsd/bin/sh/sh.full.meta
CMD cc -O2 -pipe -DSHELL -I. -I/root/git/freebsd/bin/sh -g -std=gnu99 -fstack-protecto
r-strong -Wsystem-headers -Werror -Wall -Wno-format-y2k -Wno-uninitialized -Wno-pointe
r-sign -Wno-empty-body -Wno-string-plus-int -Wno-unused-const-variable -Wno-tautologic
al-compare -Wno-unused-value -Wno-parentheses-equality -Wno-unused-function -Wno-enum-
conversion -Wno-unused-local-typedef -Wno-switch -Wno-switch-enum -Wno-knr-promoted-pa
rameter -fcolor-diagnostics -Qunused-arguments  -o sh.full alias.o arith_yacc.o arith_
yylex.o cd.o echo.o error.o eval.o exec.o expand.o histedit.o input.o jobs.o kill.o ma
il.o main.o memalloc.o miscbltin.o mystring.o options.o output.o parser.o printf.o red
ir.o show.o test.o trap.o var.o builtins.o nodes.o syntax.o   -ledit
CWD /usr/obj/root/git/freebsd/bin/sh
TARGET sh.full
-- command output --

-- filemon acquired metadata --
# filemon version 5
# Target pid 63370
# Start 1465173818.791066
V 5
E 66535 /bin/sh
R 66535 /etc/libmap.conf
R 66535 /usr/local/etc/libmap.d
R 66535 /var/run/ld-elf.so.hints
```

WITH_META_MODE

A working incremental buildworld

- Uses bmake's *meta mode* with *filemon(4)*
- Captures dependencies missing from the build
  - csu
  - libcompiler_rt
  - tools
- Skips cleaning for **make buildworld** (essentially default –*DNO_CLEAN)*
- No *.depend.target.o* generated (mostly redundant)
  - Doesn't invoke the *OBJS_DEPEND_GUESS* mechanism since it also considers a *.meta* file to be present before adding the extra dependency in

WITH_META_MODE

New output

- Uses more terse build output borrowed from *WITH_DIRDEPS_BUILD*
  - *Building /usr/obj/usr/src/lib/libclang_rt/ubsan_standalone/sanitizer_libignore.o*
  - See */usr/obj/usr/src/lib/libclang_rt/ubsan_standalone/sanitizer_libignore.o**.meta*** for build command
- Errors can show the *.meta* file used but disabled currently

```
bin/sh # make CFLAGS.exec.c=error exec.o
Building /usr/obj/root/git/freebsd/bin/sh/exec.o
cc: error: no such file or directory: 'error'
*** Error code 1

Stop.
make: stopped in /root/git/freebsd/bin/sh
.ERROR_TARGET='exec.o'
.ERROR_META_FILE='/usr/obj/root/git/freebsd/bin/sh/exec.o.meta'
```

WITH_META_MODE

Stop doing redundant things

- Usually build targets are *.PHONY* meaning they produce no file/cookie
- Can be used, along with a target cookie, to prevent a target from running again if not needed
- A lot of opportunity to do this in **make buildworld** for WORLDTMP staging for *install* targets
  - Not yet done, but the pattern is used for *WITH_DIRDEPS_BUILD*
- Normally a cookie on an *install* target is not safe…

WITH_META_MODE

Current code (safe, no meta)

```
do-install:
  install ${FILES} ${WORLDTMP}
```

Using a cookie (unsafe, no meta)

```
do-install:
  install ${FILES} ${WORLDTMP}
  touch do-install
```

WITH_META_MODE

Meta mode cookie

```
do-install:
  rm –f do-install
  install ${FILES} ${WORLDTMP}
  touch do-install
```

- Meta mode / filemon will detect if the command needs to rerun
- Must remove old cookie in case further commands fail, to try again later

Simpler

```
META_TARGETS+= do-install
do-install:
  install ${FILES} ${WORLDTMP}
```

Special case, if target defined after bsd.sys.mk

```
META_TARGETS+= do-install
do-install: ${META_DEPS}
  install ${FILES} ${WORLDTMP}
```

WITH_META_MODE

Final word

- Can be overly aggressive but generally still better than a build that does **make cleanobj** and rebuilds everything
- Initial *CFT* had some issues that are fixed now
  - **make cleanworld** no longer needed
  - **make installworld** no longer causes next build to rebuild everything
  - Significant performance improvements for *realpath(3)* handling
- 8 minute NOP build
- Not compatible with *WITH_SYSTEM_COMPILER* yet
  - -target and –sysroot build command changes
- More bmake performance improvements coming
- Bug to fix with libraries relinking
- Only for building 11+ but stable/10 will be able to use it as a build host after MFCs
- Use **–dM** flag to make to show why something is rebuilt

Miscellaneous

A lot of little things

- Build-time assertions for adding new libraries correctly
- Various bitrot cleanup
- *bsd.progs.mk* is now parallel safe and reliable
  - Interaction with *FILES*, *SCRIPTS*, *TESTS*, etc, was either skipping targets or running multiple times
- Error if installing a file to a missing directory
  - Such as installing *foo.h* into */usr/include/dest* where *dest* does not exist.
  - Simple fix (install with trailing /) but very impactful when forgetting to run **make distrib-dirs** after pulling in an updated mtree file when building/installing from a subdirectory
- More CXX support (LIB_CXX, PROG_CXX)
- *bsd.compiler.mk* compiler version caching to sub-makes

Miscellaneous

Continued

- (ACFLAGS|CFLAGS|CXXFLAGS).SRC
  - *CFLAGS.file.c= -Wspecial-flag*
- Error if building during install-time (src only)
  - *CFLAGS+= ERROR-tried-to-rebuild-during-make-install*
  - Policy to support read-only object directories and avoid very obscure **installworld** failures on various timestamp changes
- **make analyze** (from NetBSD)
  - Runs the clang static analyzer for the directory
  - No kernel support yet, only userland and modules
- *WITHOUT_CROSS_COMPILER*, *WITHOUT_TOOLCHAIN* both fixed to work
- External toolchain support simplified and expanded a bit
- *SUBDIR_PARALLEL* expanded a lot (such as all of sys/modules)

Planned Improvements

Planned improvements

- Build clang once for **make universe** if *WITH_SYSTEM_COMPILER* is not satisfied and use it for all architectures.
- Add external clang xtoolchain packages as right now there are only GCC packages
- *WITH_AUTO_OBJ*
- More build-time assertions
- Cleanup
  - Cleaning up redundancy with *Makefile.inc1 lib/dir__L* targets
  - Cleaning up redundancy with *bsd.incs.mk/bsd.files.mk/bsd.confs.mk*
  - Cleaning up redundancy with *_DP_** in src.libnames.mk
- Handbook section on the build and Journal articles

Over/Under-linked library testing

Link only everything needed

- Libraries should link in all of their own library dependencies and nothing they don't need
- Isilon's build checks for both of these cases
- Overlink
  - tools/build/check-links.sh
  - Compares **nm** output to linked libraries **nm** output
- Underlink
  - Linking libraries with *-Wl,--no-undefined*
  - Requires that all symbols used be resolved at linktime
  - Special cases which get a free pass
    - It does break the idea for "modules" that get their symbols from their loading consumers
    - Does not work if a library has a cyclic dependency with another

WISHLIST

Cross builds

Review

- Must build host tools to run during the build
- Cannot run target binaries in the build
- Need to ensure the target binaries are built with the proper libc and knowing what functions the target supports

Cross-OS builds

We can do it

- Building currently only supported from FreeBSD with *MSR* but there is demand for Linux/OSX
- Requires even more bootstrapping for early build tools such as **mtree** or **ln** (for –h) in *install.sh* or **install** which are not part of an external toolchain
- NetBSD supports this by a large compatibility library
- Requires investment and maintenance into a much larger *libegacy* for [Free]BSDisms
  - sys/cdefs.h from tree since so many headers use definitions from it, like *_Thread_local* for *xlocale.h* for *localedef*
  - FreeBSD *libc*
  - *strvis(3) …*
- The clang build requires C++11 support and falls back to GCC 4.2 if not available which breaks assumptions in the build about amd64 using clang.
  - Meaning an external compiler will be needed unless we bootstrap to the point of supporting C++11 with multiple clang versions in-tree

Ports cross-build

Isilon's need

- Isilon builds OneFS from FreeBSD where QEMU doesn't make sense since it is the same arch, just a different ABI
  - Need to run binaries during the build
- Currently our build is: buildworld -> ports -> chroot(delayedworld)
  - Run OneFS binaries from FreeBSD using a kernel module for syscall compatibility
  - We have reasons for not building from OneFS

Ports cross-build

Overview

- Currently *pkg.freebsd.org* provides arm and mips packages that are built from QEMU on amd64
- Ports metadata provides dependencies for each phase:
  - FETCH, EXTRACT, PATCH, BUILD, RUN, LIB
- Ports install to a staging directory before being packaged or installed to /
  - Some BUILD/RUN dependencies are actually used for staging and mislabeled
    - Ruby/Python ran to stage
  - Might need a STAGE_DEPENDS
- Host
  - Must always build for host FETCH, EXTRACT, PATCH dependencies
  - Must guess and build BUILD, RUN dependencies as well
- Target
  - For target only BUILD, LIB and RUN dependencies are needed
- This means some ports will build the same thing twice (like clang in base)

Ports cross-build

Doing it with DIRDEPS

- Need a unified build for Isilon
- Allows base build to depend on ports in a single dependency graph
- Ports has no parallel build mechanism (requires Poudriere)
- Using *DIRDEPS* for this with *host* and *target* staging vs. using / for *host*
- Replaces the ports dependency framework with DIRDEPS

Ports cross-build

Problems with DIRDEPS approach

- Using a target/host staging directory not likely feasible
  - *LOCALBASE* vs *HOSTBASE*
    - *LOCALBASE=/usr/local*
      - where things are already installed
    - *PREFIX=/usr/local*
      - where things should be installed
    - *HOSTBASE=/hoststage*
      - Host tools to run for the build, with *LD_LIBRARY_PATH* set
    - PKG_BIN=LOCALBASE/sbin/pkg-static
    - Perl/Python/Ruby installation directories hardcoded
  - Likely needs more of a Poudriere-style build
- Using DIRDEPS (or any parallel build) with ports is wonky with **make –j**
  - Base A –j (fail), Port A (long run but fails at end), Base B –j (notices failure immediately)

Ports cross-build

Case-by-case

- Autoconf builds require a *config.site* for the target system
  - --host=ARCH-OS (mips64-FreeBSD11.0)
  - Many tests run in the build to see if a function “works”
  - Tests that *run* during the build need **ac_cv_functionworks** overrides since autoconf assumes it can run binaries it compiles and that binaries it runs match the target
- Perl requires a config.sh or crossperl or ssh to target
- Python claims “only Linux” cross building
- Easiest method of generating these is to do so on the target and store the *config.site* in the tree
- Some ports like devel/gettext build their own build tools that are ran in the build
  - Requires building devel/gettext for host but also patch the target build to run the binaries from the host version of the build
- Not all build frameworks support cross-build

Ports cross-build

Final word

- Largely impractical
- Ports has no bootstrap mechanism
- Possible to get ports cross-built case-by-case but not all 24,000 of them
- QEMU and more powerful machines are more feasible

QUESTIONS?

- WITH_FAST_DEPEND: https://svnweb.freebsd.org/changeset/base/290433
- WITH_CCACHE_BUILD: https://svnweb.freebsd.org/changeset/base/290526
- WITH_META_MODE: https://lists.freebsd.org/pipermail/freebsd-current/2016-May/061481.html
- WITH_SYSTEM_COMPILER: https://lists.freebsd.org/pipermail/freebsd-current/2016-May/061376.html
- DIRDEPS: http://www.crufty.net/sjg/blog/freebsd-meta-mode.htm