FreeBSD building with bmake

Simon J. Gerraty
Juniper Networks, Inc.
BSDCan 2014

*Imagine something very witty here*

**Agenda**

Introduction
Why *bmake*
FreeBSD transition to using *bmake*
*Meta mode* and *dirdeps.mk* (or why *bmake* is so cool)
projects/bmake

**Introduction**

- *bmake* started 1993, derived from NetBSD make
  - with *autoconf* builds on almost everything (AIX ... UTS)
- Junos built with *bmake* since 2000
  - lots of features added to support Junos build
  - most used within NetBSD build shortly after
- Initial discussions at BSDCan 2011
- First commit of *bmake* to FreeBSD head in October 2012
- Portmgr gave green-light May 2013
  - delay due to ports infrastructure rebuild
- FreeBSD 10 uses *bmake* as */usr/bin/make*
- FreeBSD 9.3 will build and install */usr/bin/bmake* for ports
- Much thanks to FreeBSD and NetBSD projects.

**Why *bmake***

- actively maintained by multiple developers
- lots of very cool features
  - a plethora of variable modifiers (eg. :@ in-line loops)
  - complex sets of modifiers can be set in variables for easy re-use
  - multiple iterator variables in .for loops
  - auto-ignore stale dependencies from .depend
  - meta mode
- *dirdeps.mk* probably uses every single feature

BSDCan 2014
Why *meta* mode and/or *dirdeps.mk*?

- can use *meta* mode and *dirdeps.mk* independently
- simple, reliable and maintainable
  - top-level makefiles at least an order of magnitude fewer lines
  - build works the same from anywhere in the tree
  - reduces reliance on humans to get things right
- supportable
  - many developers do not log their build
  - when something goes wrong, no data to analyze
  - *.meta* files, usually provide sufficient clue

**Teaser**

Building `/bin/sh` in a clean tree:

```bash
$ time mk -j8 -C bin/sh
[Creating objdir /c/sjg/obj/projects-bmake/amd64.amd64/bin/sh...]
Checking /c/sjg/work/FreeBSD/projects-bmake/src/pkgs/pseudo(stage for amd64,amd64 ...
... Checking /c/sjg/work/FreeBSD/projects-bmake/src/include for amd64,amd64 ...
[Creating objdir /c/sjg/obj/projects-bmake/amd64.amd64/include...]
Checking /c/sjg/work/FreeBSD/projects-bmake/src/include/xlocale for amd64,amd64 ...
Checking /c/sjg/work/FreeBSD/projects-bmake/src/lib/csu/amd64 for amd64,amd64 ...
Checking /c/sjg/work/FreeBSD/projects-bmake/src/lib/libc for amd64,amd64 ...
[Creating objdir /c/sjg/obj/projects-bmake/amd64.amd64/lib/libc...]
Building /c/sjg/obj/projects-bmake/amd64.amd64/lib/libc/.dirdep
... Building /c/sjg/obj/projects-bmake/amd64.amd64/lib/libc/stdio.o
Building /c/sjg/obj/projects-bmake/amd64.amd64/lib/libc/libc.a
Building /c/sjg/obj/projects-bmake/amd64.amd64/lib/libc/stage_libs
... Checking /c/sjg/work/FreeBSD/projects-bmake/src/bin/sh/Makefile.depend: .depend token.h.meta builtins.c.meta mknodes.o.meta mksyntax.o.meta sh.1.gz.meta mksyntax.meta mknodes.meta
58.02 real 204.38 user 72.80 sys
```

*it's hard to make a build log interesting.*

**Teaser cont...**

```bash
Building /c/sjg/obj/projects-bmake/amd64.amd64/bin/sh/syntax.o
Building /c/sjg/obj/projects-bmake/amd64.amd64/bin/sh/sh
Building /c/sjg/obj/projects-bmake/amd64.amd64/bin/sh/.dirdep
Building /c/sjg/obj/projects-bmake/amd64.amd64/bin/sh/stage_as.prog
Checking /c/sjg/work/FreeBSD/projects-bmake/src/bin/sh/Makefile.depend: .depend token.h.
```

**Things to note:**

- objdirs were created automatically
- no *make* depend
- everything ran in parallel, but in the correct order

BSDCan 2014
• log easy to read - generally only single line per target
• only built that which was necessary
• leaf dirs visited directly - no tree walks
• Makefile.depend

A quick look at Makefile.depend

```bash
# Autogenerated - do NOT edit!

DEP_RELDIR := ${_PARSEDIR:S,${SRCTOP}/,,}

DIRDEPS = \n  gnu/lib/libgcc \n  include \n  include/xlocale \n  lib/$(CSU_DIR) \n  lib/libc \n  lib/libcompiler_rt \n  lib/libedit \n  lib/ncurses/ncurses 

.include <dirdeps.mk>

.if $(DEP_RELDIR) == ${_DEP_RELDIR}
# local dependencies - needed for -jN in clean tree
arith_yylex.o: syntax.h
...
.endif
```

Building FreeBSD

• projects/bmake is test case for generic meta.*.mk and dirdeps.mk
• want to be able to easily cross-build stock FreeBSD
• minimize changes to FreeBSD
• Juniper FreeBSD team build head and stable/10 in meta mode, much same as projects/bmake
  • external toolchains
  • generate packages (isofs images mostly)
  • disk images for booting VM

FreeBSD transition to using bmake

• base
  • reasonably simple (< 300 line diff)
• ports
  • complicated by need to support older FreeBSD without branching

BSDCan 2014
• other FreeBSD users WITH[OUT]_BMAKE
  • not everyone wants to be a pioneer
  • changes made to bmake for FreeBSD
  • changes are made in NetBSD where possible

**bmake vs fmake**

• both descended from pmake but have diverged significantly
• FreeBSD make (fmake) has :U and :L modifiers that conflict
  • not used by base
  • used by ports
• bmake uses .PATH aggressively, requires explicit .NOPATH in some cases. Generally:

```
.NOPTH: $(CLEANFILES)
```

**bmake vs fmake cont...**

• NetBSD’s bsd.own.mk flags all standard targets as .PHONY and .NOTMAIN
• handling of job tokens
  • fmake uses FIFO with name exported to sub-makes
  • bmake uses pipe with descriptors passed to sub-makes (.MAKE)
  • each works fine, but do not mix well
• .info very handy (neater than x!= echo blah >&2; echo)

**Changes to base**

• protect bmake specific syntax with:

```
.if defined(.PARSEDIR)
  # bmake
```

• add .NOPATH for generated files
• add .PHONY and .NOTMAIN for standard targets
• add .WAIT as no-op target for fmake
• add BMAKE option (WITH[OUT]_BMAKE)

**Changes for base - BSDmakefile**

• fmake looks for BSDmakefile makefile Makefile
• bmake normally only looks for makefile Makefile
• bmake is configurable (in sys.mk):

BSDCan 2014
# Tell bmake the makefile preference

.TMAKE=BSDmakefile makefile Makefile

Changes for base - job tokens

- **bmake** only passes job token descriptors to targets flagged with .MAKE
- **fmake** doesn't do .MAKE correctly, so not widely used (hence: $(_+_))
- add local hack to pass job token descriptors to all children
  - **unless** .MAKE.ALWAYS_PASS_JOB_QUEUE=no

Changes for base - error token

Normally when a sub-make fails, bmake puts an error token into the job token pool. This causes all other sub-makes to quickly spot the failure and bail.

This is exactly what you want - usually.

But not for make universe, so if MAKE_JOB_ERROR_TOKEN is false, no error token is pushed into pool.

Changes for base - errCheck

- **fmake** runs target scripts with `set -e` this means that target fails if any statement within a command line fails:


```bash
  cd /nowhere; rm -rf *
```

- **bmake** runs target scripts such that the command line (rather than individual statements) must fail:


```bash
  (cd /nowhere && rm -rf *)
```

- the later is safe with any version of make

Changes for base - errCheck cont...

Resolved by adding to sys.mk:

```
# By default bmake does *not* use set -e
# when running target scripts, this is a problem for many makefiles here.
# So define a shell that will do what FreeBSD expects.
ifndef WITHOUT_SHELL_ERRCTL
  SHELL: name=sh \
  quiet="set -" echo="set -v" filter="set -" \ 
  hasErrCtl=yes check="set -e" ignore="set +e" \ 
  echoFlag=v errFlag=e \ 
  path=${__MAKE_SHELL:U/bin/sh}
.endif
```

### Option **BMAKE** or WITH[OUT]_BMAKE

BSDCan 2014
• initial proposal add /usr/bin/bmake
  • install fmake as /usr/bin/fmake
  • /usr/bin/make a symlink
• plan agreed was to switch /usr/bin/make ASAP
• even if base could cut-over, others had concerns/issues
  • add option BMAKE initially default NO, later YES
  • src/Makefile needs to DTRT for WITH[OUT]_BMAKE
  • use temp make named fmake or bmake as needed
  • WITHOUT_BMAKE support recently removed from head

Ports - :L and :U

• ports uses :L and :U until 8.3 EOL
  • 8.4 and later support :tl and :tu
  • add temp local hack for bsd.port.mk:
    
    # tell bmake we use the old :L :U modifiers
    .MAKE.FreeBSD_UL= yes

    • no longer needed - ports converted to :tl and :tu
  • quoted strings as .for loop iterators
• added to NetBSD

Ports -v behavior

• fmake -V FOO prints fully resolved value
• bmake -V FOO prints literal value (${DESTDIR}/path/to/foo ?)
  • bmake -V '${FOO}' gives resolved value
  • great for debugging - not necessarily ideal from build pov
• Added knob .MAKE.EXPAND_VARIABLES to select behavior
• Added debug flags -dV to print literal value regardless

Ports MLINKS loops

• bmake substitutes $(:,Uvalue) for iterators
• nested .for loops with escaped iterators to handle MLINKS replace with:

  .if defined(.PARSEDIR)
  # inline loops are simpler
  _MLINKS= $(._MLINKS_PREPEND) \
  ${MANLANG:S,^,man/,:S,/*",,:@m${MLINKS:@p$${MAN$(p:E)PREFIX}}/m/man${p:E}/$p${MANEXT}@}@}
  .else

  BSDCan 2014
**Why bmake is so cool**

- It can do `dirdeps.mk` (see contrib/bmake/mk/dirdeps.mk)
- Some useful modifiers:

```plaintext
CXXSEED ?= -frandom-seed=${.ALLSRC:T:O:u:hash}

# a handy token
_this = ${.PARSEDIR:tA}/${.PARSEFILE}

# We use this a lot, it turns a list into a set of :N modifiers
# Eg.
# NskipFoo= ${Foo:${M_ListToSkip}}
M_ListToSkip = O:u:ts::S,,:,N,g:S,^,N,

TIME_STAMP_FMT ?= @ %s [\%Y-%m-%d %T]
TIME_STAMP = $(TIME_STAMP_FMT:localtime)

$ make LIST="one two three" -V '${LIST:${M_ListToSkip}}' -V TIME_STAMP
None:Nthree:Ntwo
@ 1399523065 [2014-05-07 21:24:25]
```

**projects/bmake**

- last sync'd from head May 8
- generic `Makefile.depend`, very few `Makefile.depend.$(MACHINE)`
- pkgs/Makefile acts as top-level
- pkgs/pseudo/*/Makefile.depend provide build targets:

```plaintext
pkgs/pseudo/bootstrap-tools/Makefile.depend.host
pkgs/pseudo/toolchain/Makefile.depend
pkgs/pseudo/userland/Makefile.depend
pkgs/pseudo/kernel/Makefile.depend
```

**projects/bmake environment**

The `mk` reads `.sandbox-env` at top of tree (it searches upwards for it) to condition the environment. Sets `SB` to the directory where `.sandbox-env` found. The critical value is:

```plaintext
export MAKESYSPATH="$SB/src/share/mk"
```

It sets others which `local.sys.mk` could do if needed:

```plaintext
export SRCTOP=$SB/src
export OBJROOT=$SB/obj/
export MAKEOBJDIR='${CURDIR:S,${SRCTOP},${OBJTOP},}'
```

OBJTOP is set by `local.sys.mk` to $(OBJROOT)$(MACHINE)

BSDCan 2014
projects/bmake environment cont...

Since most setup can be done via `local.sys.mk` One could just set:

```bash
export MAKESYSPATH=.../share/mk
```

in `~/.profile` or `~/.login` and `bmake` would DTRT

projects/bmake status

- build all? userland, kernel, toolchain (for host and target)
- added `pkgs/pseudo/bootstrap-tools` to help transition to new clang
  - simply leverages targets from `src/Makefile.incl`
  - now using sysroot
  - can buildworld while producing `.meta` files
    - something to compare against
    - can help with bootstrapping `Makefile.depend`

projects/bmake getting started

environment:

```bash
export MAKESYSPATH=.../share/mk
export MAKEOBJDIR='${.CURDIR:S,${SRCTOP},${OBJTOP},}'
```

bootstrap tools:

```bash
make -C pkgs -j8 bootstrap-tools
MACHINE=host make -C pkgs -j8 toolchain -DWITH_TOOLSDIR
```

have fun:

```bash
make -j8 -C bin/sh
make -j18 -C pkgs the-lot
```

depending build failure

Create a failure - add a bogus include to `bin/cat/cat.c` and compile:

```
$ make -j8 -C bin/cat -DNO_DIRDEPS
[Creating objdir /c/sjg/obj/projects-bmake/amd64.amd64/bin/cat...]
Checking /c/sjg/work/FreeBSD/projects-bmake/src/bin/cat for amd64,amd64 ... 
Building /c/sjg/obj/projects-bmake/amd64.amd64/bin/cat/cat.o
Building /c/sjg/obj/projects-bmake/amd64.amd64/bin/cat/cat.1.gz
--- cat.o ---
/c/sjg/work/FreeBSD/projects-bmake/src/bin/cat/cat.c:351:10: fatal error: 'oops.h' file #include "oops.h"
  ^
```

BSDCan 2014
1 error generated.
*** [cat.o] Error code 1

make[1]: stopped in /c/sjg/work/FreeBSD/projects-bmake/src/bin/cat
.ERROR_TARGET='cat.o'
.ERROR_META_FILE='/c/sjg/obj/projects-bmake/amd64.amd64/bin/cat/cat.o.meta'
.MAKE.LEVEL='1'
.MAKE.MODE='meta verbose silent=yes'
.CURDIR='/c/sjg/work/FreeBSD/projects-bmake/src/bin/cat'
.OBJDIR='/c/sjg/obj/projects-bmake/amd64.amd64/bin/cat'
...
ERROR: log /c/sjg/work/FreeBSD/projects-bmake/error/meta-99526.log

debbuging build failure cont...
The failed meta file is copied to ${SRCTOP:H}/error/meta-*.log:

```
# Meta data file /c/sjg/obj/projects-bmake/amd64.amd64/bin/cat/cat.o.meta
CMD cc -O2 -pipe --sysroot=/c/sjg/obj/projects-bmake/stage/amd64.amd64/ -std=gnu99 -fstack-protector -Wsystem-headers ... -Wno-unused-const-variable -Qunused-arguments -c /c/sjg/work/FreeBSD/projects-bmake/src/bin/cat/cat.c
CMD
CWD /c/sjg/obj/projects-bmake/amd64.amd64/bin/cat
TARGET cat.o
-- command output --
/c/sjg/work/FreeBSD/projects-bmake/src/bin/cat/cat.c:351:10: fatal error: 'oops.h' file not found
#include "oops.h"
^
1 error generated.
*** Error code 1
```

```
-- filemon acquired metadata --
# filemon version 4
# Target pid 99526
# Start 1400265589.023156
...
E 99535 /c/sjg/obj/projects-bmake/stage/freebsd10-amd64/usr/bin/cc
...
X 99535 1
...
```

projects/bmake next steps

- add target to build a bootable VM image
  - makefs to create filesystems
  - mkimg to wrap them up
- distributed build?
  - dirdeps.mk makes it quite simple

bsd.* src.* and local.*

bsd.*.mk
  generic build logic

BSDCan 2014
src.*.mk
   build logic specific to building /usr/src

local.*.mk
   logic specific to a tree or site, allows customization without hacking

Only bsd.*.mk installed in /usr/share/mk

Eg. sys.mk, bsd.init.mk, dirdeps.mk ... all do:

   .-include "local.${.PARSEFILE:S,bsd.,,}"  

src.opts.mk

We can now take advantage of src.opts.mk eg. pkgs/pseudo/toolchain/Makefile.depend:

```makefile
DEP_RELDIR := ${_PARSEDIR:S,${SRCTOP}/,,}

  .if !defined(MK_CLANG)
    .include "${SRCTOP}/share/mk/src.opts.mk"
  .endif

DIRDEPS= usr.bin/xinstall
  .if ${MK_CLANG} == "yes"
    DIRDEPS+= pkgs/pseudo/clang
  .endif
  .if ${MK_GCC} == "yes"
    DIRDEPS+= pkgs/pseudo/gcc
  .endif

  .include <dirdeps.mk>
```

Automated .OBJDIR creation

With bmake, makefiles can control .OBJDIR, this makes automated objdir creation possible (from auto.obj.mk):

```makefile
  .if !defined(NOOBJ) && !defined(NO_OBJ) && ${MKOBJDIRS:Uno} == auto
  # Use __objdir here so it is easier to tweak without impacting
  # the logic.
  __objdir?= ${MAKEOBJDIR}
  .if ${.OBJDIR} != ${__objdir}
    # We need to chdir, make the directory if needed
    .if !exists($(__objdir)/) && 
      ($(.TARGETS) == "" || ${.TARGETS:Nclean*:N*clean:Ndestroy:*} != "")
      # This will actually make it...
      __objdir_made != echo $(__objdir)/; umask ${OBJDIR_UMASK:U002}; \
        ${ECHO_TRACE} "[Creating objdir $(__objdir)...]" >&2; \
            ${Mkdirs}; Mkdirs ${__objdir}
    .endif
    # This causes make to use the specified directory as .OBJDIR
    .OBJDIR: ${__objdir}
    .if ${.OBJDIR} != ${__objdir} && __objdir_made:Uno: M$(__objdir)/" != ""  
      .error could not use $(__objdir)
```

BSDCan 2014
Introducing *Meta Mode*

- create a `.meta` file for each target
- `.meta` file collects information about the target
  - the expanded command line
  - command output
  - *interesting* system calls

**Rationale**

- aid automated capture of dependency information
  - help optimize build performance
  - improve build reliability
- optimizing build means
  - do as little as possible
  - do it in parallel
  - but do it correctly!
- capture command output
  - makes failure analysis feasible
- *meta* mode and `dirdeps.mk` help all the above

**avoid make depend**

- saves a lot of time
- requires better makefiles for parallel building
  - capture local dependencies to `Makefile.depend` for clean tree build
- `filemon` works for all targets not just `gcc`
  - automatically catches toolchain changes

**avoid unnecessary dependencies**

- In *meta* mode, `bmake` can compare expanded commands to *know* if there is a change. Thus dependencies like:

```bash
# if any of the makefiles have changed we need to regenerate
# this - "just in case"
generated.h:    ${.MAKE.MAKEFILES:N.depend}
${OBJJS}:        generated.h
```

can be skipped.
- can use `DPADD` to bootstrap `DIRDEPS`
- entries in `DPADD` but not `DIRDEPS` were unnecessary.

BSDCan 2014
some targets may need work
Some targets need attention to avoid always being out-of-date.
For example if we do not want version.c regenerated every time (because ${TIME_STAMP} changed):

```
version.c: .NOMETA_CMP
  @echo 'static char id[] = "@(#) build $(TIME_STAMP) by $(USER)"'; > $(.TARGET)
```

Same for versionxtra.c but we do want to regenerate if ${DPADD} changes:

```
versionxtra.c:
  @echo 'static char id[] = "@(#) built $(TIME_STAMP) by $(USER)"'; > $(.TARGET) \
  ${.OODATE:M.NOMETA_CMP}
  @echo 'static char libs[] = "${DPADD}"'; >> $(.TARGET)
```

Building in meta mode

- enabled by the word meta in .MAKE.MODE which can be set by makefile
- makefiles can set .MAKE.MODE = normal to avoid meta mode.
- meta.sys.mk included by sys.mk, does:

```
.if ${.MAKE.LEVEL} == 0
# make sure dirdeps exists and do it first
all: dirdeps .WAIT
dirdeps:
.endif
META_MODE += meta verbose
.MAKE.MODE ?= $(META_MODE)
```

Writing .meta files

- for each target, a .meta file called ${.TARGET}.meta is created
- if target is .PHONY, .MAKE or .SPECIAL (eg. .BEGIN, .END, .ERROR), then a .meta file is not created unless the target is also flagged .META
- never created if target flagged .NOMETA
- skip .meta if .OBJDIR == .CURDIR and curdirOk=yes not in .MAKE.MODE
- if target not in ${.OBJDIR}, replace all / with _ in meta file name

Meta file content

- expanded command line(s), prefixed with CMD
- current directory prefixed with CWD
- target, prefixed with TARGET
- command output preceded by line -- command output --
  - this is useful for error handling

BSDCan 2014
syscall data collected from filemon preceded by line -- filemon acquired metadata --
append the name of the .meta file to variables .MAKE.META.CREATED and .MAKE.META.FILES
if meta verbose mode expand and print .MAKE.META.PREFIX which defaults to the full path of the target.

filemon

- kernel module replaces use of DTrace
- available in FreeBSD, NetBSD and Linux
- for each syscall, an entry of the form:

<table>
<thead>
<tr>
<th>tag</th>
<th>data</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>chdir</td>
</tr>
<tr>
<td>D</td>
<td>unlink</td>
</tr>
<tr>
<td>E</td>
<td>exec</td>
</tr>
<tr>
<td>F</td>
<td>[v]fork</td>
</tr>
<tr>
<td>L</td>
<td>[sym]link</td>
</tr>
<tr>
<td>M</td>
<td>rename</td>
</tr>
<tr>
<td>R</td>
<td>open for read</td>
</tr>
<tr>
<td>S</td>
<td>stat</td>
</tr>
<tr>
<td>W</td>
<td>open for write</td>
</tr>
<tr>
<td>X</td>
<td>exit</td>
</tr>
</tbody>
</table>

- bmake mainly interested in C E L M and R entries

Reading .meta files

- skipped if target already considered out-of-date
- use -dM to see why bmake thinks target out-of-date
- compare expanded commands
  - unless told not to (.NOMETA_CMP)
  - or commands use ${.OODATE} (hint: ${.OODATE:M.NOMETA_CMP})
- compare mtime of files Read or Executed against target
- if generated file within ${.MAKE.META.BAILIWICK} but outside ${.OBJDIR} is missing, target is out-of-date

Extracting dependencies

- bmake simply uses .meta files to better know when a target is out-of-date
- bmake tracks .meta files via .MAKE.META.FILES and .MAKE.META.CREATED
- allows makefiles such as meta.autodep.mk to post-process .MAKE.META.FILES to gather tree wide dependencies.
- this process is greatly simplified by keeping objdirs out of the src tree
post-processing meta files

```
# Meta data file /c/sjg/work/FreeBSD/current/obj/i386/bin/sh/var.o.meta
...  
-- filemon acquired metadata --
...
E 16111 /bin/sh
...
R 16112 /c/sjg/work/FreeBSD/current/src/bin/sh/var.c
W 16113 var.o
R 16112 /c/sjg/work/FreeBSD/current/obj/stage/i386/usr/include/sys/cdefs.h
R 16112 /c/sjg/work/FreeBSD/current/obj/stage/i386/usr/include/unistd.h
...  
R 16112 /c/sjg/work/FreeBSD/current/obj/stage/i386/usr/include/unistd.h
R 16112 /c/sjg/work/FreeBSD/current/obj/stage/i386/usr/include/stddef.h
R 16112 /c/sjg/work/FreeBSD/current/obj/stage/i386/usr/include/expand.h
R 16112 ./nodes.h
```

- any file read or executed from an objdir other than .OBJDIR identifies a directory which must be built before .CURDIR, (DIRDEPS)
- any file read from the the src tree outside of .CURDIR identifies a directory which must exist, (SRC_DIRDEPS)

mapping objdir to src dir

- when linking libraries from their objdir, the mapping to src dir is trivial:

  ```
  SRC_libfoo = ${OBJ_libfoo:S,${OBJTOP},${SRCTOP},}
  ```

- when using headers and libraries which have been staged, help is needed:

  ```
  $ cd /c/sjg/work/FreeBSD/current/obj/stage/i386/usr/include
  $ ls -l unistd.h*
  -rw-r--r-- 2 sjg wheel 18731 Mar  2 18:37 unistd.h
  -rw-r--r-- 92 sjg wheel  13 Apr  3 14:53 unistd.h.dirdep
  $ cat unistd.h.dirdep
  include.i386  
  ```

  the .dirdep file contains the DIRDEPS entry needed.

Logs are still useful

If we run our build of bin/sh with -DWITH_META_STATS:

```
Finished gnu/lib/csu.amd64,amd64 seconds=0 meta=7  created=0
...
Finished lib/libc.amd64,amd64 seconds=11 meta=4783  created=41
...
Finished bin/sh.amd64,amd64 seconds=0 meta=42  created=0
```

Makefiles

BSDCan 2014
• majority of leaf makefiles *just work*
• some minor changes to `bsd.*.mk`
• new makefiles `meta.*.mk`, `dirdeps.mk` and `gendirdeps.mk`
• top level makefiles can be very simple
• `Makefile.depend` is most visible change

**Makefile.depend**

• collects `DIRDEPS` and local dependencies for each directory
• can be maintained in SCM
• use `Makefile.depend.$(MACHINE)` when necessary.

**One build product per directory**

• building multiple things is ok but
• each directory/makefile should do the same thing every time
• only collect dependencies when doing default target

**meta.autodep.mk**

• post-processing `.meta` files can be expensive, skip if possible
• if `.MAKE.META.CREATED` is not empty, we have work to do
• process `.MAKE.META.FILES`:

```makefile
.END:          gendirdeps

_DEPENDFILE := $(CURDIR)/$(MAKE.DEPENDFILE:T)
gendirdeps:  $(DEPENDFILE)

# the double $$ defers initial evaluation
$(DEPENDFILE): $(MAKE.META.CREATED) $(PARSEDIR)/gendirdeps.mk
  @echo Updating $@: ${OODATE:T:[1..8]}
  (cd $(CURDIR) &
   SKIP_DIRDEPS='$(SKIP_DIRDEPS:O:u)' \
   $(MAKE) __objdir=$(OBJDIR) -f gendirdeps.mk $@
   META_FILES='$(MAKE.META.FILES:T:O:u)' )
```

gendir deps.mk

• runs `meta2deps.sh` or `meta2deps.py` to extract *interesting* directories
• things in `${SRCTOP}/*` are `SRC_DIRDEPS`
• things in `${OBJTOP}/*` are `DIRDEPS`
• things in objdirs other than `${OBJTOP}` (ie. build for other `${MACHINE}`) are qualified `DIRDEPS`.

BSDCan 2014
meta.stage.mk

- links or copies files into staging locations (think auto-install)
- puts .dirdep file next to each staged file, so mapping to src directory not lost
- multiple STAGE_SETS with own STAGE_DIR
- STAGE_AS_SETS for renaming while staging
- provides various simple targets stage_incs, stage_libs, stage_symlinks and generic stage_files and stage_as_files

dirdeps.mk

- deals with DIRDEPS
- only interesting to initial instance of bmake ($(MAKELEVEL) == 0)
- conceptually simple (< 250 lines not counting comments)
  - initial bmake reads $(CURDIR)/Makefile.depend[.$(MACHINE)] gets DIRDEPS
  - generate dependencies on each $(DIRDEP) for $(DEP_RELDIR)
  - process Makefile.depend* from each $(DIRDEP)
  - repeat

dirdeps.mk cont.

Given:

```
DIRDEPS = lib/libc include ...
```

then (ignoring the complication of other machines):

```
# always qualified
_build_dirs := $(DIRDEPS:@d@$(SRCTOP)/$d.$(MACHINE)@)

$(SRCTOP)/$(DEP_RELDIR).$(MACHINE): $(_build_dirs)

.for f in $(_build_dirs:@d@$(d:R)/$(MAKEDEPENDFILE:T)@)
 .if $(MAKE.MAKEFILES:M$(f)) == ""
  .-include <$f>
 .endif
.endfor
```

Suppressing DIRDEPS

Use -DNO_DIRDEPS to suppress DIRDEPS outside of .CURDIR:

```
$ mk-host -DNO_DIRDEPS -C external/bsd/atf/tests
```

builds and runs all unit tests in that subtree without checking anything else.

BSDCan 2014
Building kernels

- BSD kernel build does not provide a src dir per kernel to capture dependencies
- jnx.kernel.mk lets us build kernels anywhere:

```bash
# for each kernel we have:
# ${KERNEL_NAME}/config/
# ${KERNEL_NAME}/kernel/
# and possibly?
# ${KERNEL_NAME}/modules/*
#
# config/ is where config(8) is run
# both kernel/ and modules that need to link with it
# can depend on config/
# If there are kernel specific modules (which do not link into it)
# they could be built under modules/ (one directory each of course)

# Because config(8) produces a Makefile which we want to use,
# the makefiles in config/ and kernel/ above should be called 'makefile'.
```

Building kernels cont...

In projects/bmake we have pkgs/pseudo/kernel/ which can build any kernel (default GENERIC):

```bash
# Build the kernel ${KERNCONF}
KERNCONF?= ${KERNEL:UGENERIC}
TARGET?= ${MACHINE}

# keep this compatible with peoples expectations...
KERN_OBJDIR= ${OBJTOP}/sys/compile/${KERNCONF}
KERN_CONFDIR= ${SRCTOP}/sys/${TARGET}/conf
CONFIG= ${STAGE_HOST_OBJTOP}/usr/sbin/config

${KERNCONF}.config: .MAKE .META
    mkdir -p ${KERN_OBJDIR:H}
    (cd ${KERN_CONFDIR} && 
     ${CONFIG} ${CONFIGARGS} -d ${KERN_OBJDIR} ${KERNCONF})
    (cd ${KERN_OBJDIR} && ${.MAKE} depend)
    @touch $@
```
kernels cont...

```bash
# we need to pass curdirOk=yes to meta mode, since we want .meta files
# in ${KERN_OBJDIR}
${KERNCONF}.build: .MAKE $(KERNCONF).config
  (cd ${KERN_OBJDIR} && META_MODE="${.MAKE.MODE} curdirOk=yes" $(.MAKE))

.if ${.MAKE.LEVEL} > 0
all: ${KERNCONF}.build
.endif

UPDATE_DEPENDFILE= no
.include <bsd.prog.mk>
```

Top-level makefiles?

Given a collection of directories `pkgs/*` that contain little more than `Makefile.depend*`, the top-level makefile need be no more complex than:

```bash
DIRDEPS = ${.TARGETS:Nall:0d@pkgs/$d@}
.include <dirdeps.mk>

.for t in ${.TARGETS:Nall}
  $t: dirdeps
  .endfor
```

Staging headers and libs

- like `make install` as you go
- no need to be root
- minor changes to `bsd.lib.mk, bsd.incs.mk` to leverage `meta.stage.mk`

Debugging

- `bmake -dM` will say why `meta` mode decides out-of-date
- `sys.mk` supports enabling make flags in certain dirs:

```
DEBUG_MAKE_FLAGS=-dM DEBUG_MAKE_DIRS='*/libc' mk
```

Conclusion

While `meta` mode may be the coolest thing since sliced bread, it may not be for everyone.

It does provide a simple solution to some rather complex problems

http://www.crufty.net/help/sjg/bmake.htm
http://www.crufty.net/sjg/docs/freebsd-meta-mode.htm

BSDCan 2014
Some definitions

`.CURDIR`: the value returned by `getcwd(3)` when `make` first starts

`.OBJDIR`: the directory `make` is in when it starts building things

`MACHINE`: the specific machine or cpu that we are building for

`MACHINE_ARCH`: the architecture that matches `$(MACHINE)`

   armeb armv6 powerpc64 ...

`TARGET_SPEC`

- sometimes `$(MACHINE)` is insufficient to differentiate targets.
- FreeBSD universe target builds multiple `MACHINE_ARCH` per `MACHINE`.
- Junos builds multiple `TARGET_OS` per `MACHINE`.
- `dirdeps.mk` constructs `TARGET_SPEC` from `TARGET_SPEC_VARS`

`TARGET_SPEC_VARS`

- `sys.mk` needs to set `TARGET_SPEC_VARS`
- decompose `$(TARGET_SPEC)` back into components:

```bash
# Always list MACHINE first,  # other variables might be optional.
# other variables might be optional.
TARGET_SPEC_VARS = MACHINE TARGET_OS

.tspec := $(TARGET_SPEC:Uno:M*,*) != ""
MACHINE := ${tspec[1]}
TARGET_OS := ${tspec[2]}
# etc.
```

`.OBJDIR`

Make's predilection for finding an object dir causes confusion for those unfamiliar with it.

The basic algorithm is (in Bourne shell):
for __objdir in $(MAKEOBJDIRPREFIX)${.CURDIR} \\
  ${MAKEOBJDIR} \\
  ${.CURDIR}/obj.${MACHINE} \\
  ${.CURDIR}/obj \\
  ${.CURDIR}
do
  if [ -d ${__objdir} -a ${__objdir} != ${.CURDIR} ]; then
    break
  fi
done

Separating sources and objects

- default ${.CURDIR}/obj/ or ${.CURDIR}/obj.${MACHINE}/ insufficient
  - cannot do read-only src tree
  - cannot simply rm -rf ${OBJTOP}
- MAKEOBJDIRPREFIX easy - but ugly
- bmake allows applying modifiers to MAKEOBJDIR

$ export MAKEOBJDIR='${.CURDIR:S,${SRCTOP},${OBJTOP},}'

Separating sources and objects cont.

Well defined SRCTOP and OBJTOP simplify things.

One can simply assert:

CRYPTOBJDIR= ${OBJTOP}/secure/lib/libcrypt

rather than guess (wrongly):

.if exists(${.CURDIR}/..../lib/libcrypt/obj)
CRYPTOBJDIR= $(.CURDIR)/..../lib/libcrypt/obj
.else
CRYPTOBJDIR= $(.CURDIR)/..../lib/libcrypt
endif

Directory based dependencies

Allow the build to visit leaf dirs directly

- tree walks are expensive (especially on NFS)
  - may be impossible to adequately order the build steps without resorting to phases like make includes and make libraries.
- each directory should behave the same each time
- variations require separate directories

Manual maintenance is unreliable

BSDCan 2014
• not all C programmers are build geeks

basic rules for writing leaf makefiles:

1. Do not put anything in your makefile that you don't need
2. Do not put anything in your makefile that you cannot explain the need for. Ie. if you cannot explain it, you don't need it, remove it.
3. Do not cut/paste anything from your friend's makefile (see #1).

Note: #2 does not mean that you should remove everything from an existing makefile that you don't understand the first time you look at it.

• makefiles (like C code), can accrete dependencies which in many cases are unnecessary
  the less humans need to maintain, the better

meta.subdir.mk

• we do not tree walk
• may still want to launch a build in src/usr.bin/
• set initial DIRDEPS based on result of find ${SUBDIR} if no Makefile.depend* exists in .CURDIR

BUILD_AT_LEVEL0

• dependencies best met by avoiding building at level 0 controlled by BUILD_AT_LEVEL0
• no means sub-makes used to build .CURDIR even for current MACHINE
• yes means sub-makes only used to build .CURDIR for other MACHINE values.

dpadd.mk

Given:

LIBFOO ?= ${OBJTOP}/lib/libfoo/libfoo.a

If ${LIBFOO} is referenced in DPADD, dpadd.mk computes:

OBJ_libfoo = ${LIBFOO:H}
SRC_libfoo ?= ${OBJ_libfoo:S,${OBJTOP},${SRCTOP},}
  .if exists(${SRC_libfoo}/h)
  INCLUDES_libfoo ?= -I${SRC_libfoo}/h
  .else
  # all bets are off
  INCLUDES_libfoo ?= -I${OBJ_libfoo} -I${SRC_libfoo}
  .endif

dpadd.mk cont.

Since accurate dependencies in makefiles are key, we use DPLIBS:

DPLIBS += ${LIBFOO}
is equivalent to:

```
DPADD += ${LIBFOO}
LDADD += -lfoo -L${OBJ_libfoo}
```

If `${LIBFOO}` in any of SRC_LIBS, DPADD or DPLIBS:

```
CFLAGS += ${INCLUDES_libfoo}
```