NAME

build - general instructions on how to build the FreeBSD system

DESCRIPTION

The sources for the FreeBSD system and its applications are contained in three directories, normally:

/usr/src "base system", loosely defined as everything required to build the system to a useful state

/usr/doc system documentation, excluding manual pages

/usr/ports
contributed applications, with a consistent interface for building and installing them, see ports(7)

These directories may be initially empty or non-existent until updated with Git (*devel/git* from the FreeBSD Ports Collection).

The make(1) command is used in each of these directories to build and install the things in that directory. Issuing the make(1) command in any directory issues the make(1) command recursively in all subdirectories. With no target specified, the items in the directories are built and no further action is taken.

A source tree is allowed to be read-only. As described in make(1), objects are usually built in a separate object directory hierarchy specified by the environment variable *MAKEOBJDIRPREFIX*, or under */usr/obj* if variable *MAKEOBJDIRPREFIX* is not set. The canonical object directory is described in the documentation for the **buildworld** target below.

The **build** may be controlled by defining make(1) variables described in the *ENVIRONMENT* section below, and by the variables documented in make.conf(5).

The default components included in the build are specified in the file */etc/src.conf* in the source tree. To override the default file, include the SRCCONF option in the make steps, pointing to a custom src.conf file. For more information see src.conf(5).

The following list provides the names and actions for the targets supported by the build system:

analyze Run Clang static analyzer against all objects and present output on stdout.

check Run tests for a given subdirectory. The default directory used is *${.OBJDIR}*, but the check directory can be changed with *${CHECKDIR}*.

checkworld

Run the FreeBSD test suite on installed world.

clean Remove any files created during the build process.

cleandepend
Remove the *${.OBJDIR}/${DEPENDFILE}** files generated by prior "make" and "make depend" steps.

cleandir Remove the canonical object directory if it exists, or perform actions equivalent to "make clean cleandepend" if it does not. This target will also remove an *obj* link in *${.CURDIR}* if that exists.

It is advisable to run "make cleandir" twice: the first invocation will remove the canonical object directory and the second one will clean up *${.CURDIR}*.

depend Generate a list of build dependencies in file *${.OBJDIR}/${DEPENDFILE}*. Per-object dependencies are generated at build time and stored in *${.OBJDIR}/${DEPENDFILE}.${OBJ}*.

install Install the results of the build to the appropriate location in the installation directory hierarchy specified in variable *DESTDIR*.

obj Create the canonical object directory associated with the current directory.

objlink Create a symbolic link to the canonical object directory in *${.CURDIR}*.

tags Generate a tags file using the program specified in the make(1) variable *CTAGS*. The build system supports ctags(1) and **GNU Global**.

The other supported targets under directory */usr/src* are:

buildenv Spawn an interactive shell with environment variables set up for building the system or individual components. For cross-building the target architecture needs to be specified with make(1) variables *TARGET_ARCH* and *TARGET*.

This target is only useful after a complete toolchain (including the compiler, linker, assembler, headers and libraries) has been built; see the **toolchain** target below.

buildworld Build everything but the kernel, configure files in *etc*, and *release*. The object directory can be changed from the default */usr/obj* by setting the *MAKEOBJDIRPREFIX* make(1)

variable. The actual build location prefix used depends on the *WITH_UNIFIED_OBJDIR* option from src.conf(5). If enabled it is *${MAKEOBJDIRPREFIX}${.CURDIR}/${TARGET}.${TARGET_ARCH}* for all builds. If disabled it is *${MAKEOBJDIRPREFIX}${.CURDIR}* for native builds, and *${MAKEOBJDIRPREFIX}/${TARGET}.${TARGET_ARCH}${.CURDIR}* for cross builds and native builds with variable *CROSS_BUILD_TESTING* set.

cleankernel Attempts to clean up targets built by a preceding **buildkernel**, or similar step, built from the same source directory and *KERNCONF*.

cleanworld Attempt to clean up targets built by a preceding **buildworld**, or similar step, built from this source directory.

cleanuniverse When *WITH_UNIFIED_OBJDIR* is enabled, attempt to clean up targets built by a preceding **buildworld**, **universe**, or similar step, for any architecture built from this source directory.

distributeworld

Distribute everything compiled by a preceding **buildworld** step. Files are placed in the directory hierarchy specified by make(1) variable *DISTDIR*. This target is used while building a release; see release(7).

native-xtools This target builds a cross-toolchain for the given **TARGET** and **TARGET_ARCH**, as well as a select list of static userland tools for the host system. This is intended to be used in a jail where QEMU is used to improve performance by avoiding emulating binaries that do not need to be emulated. **TARGET** and **TARGET_ARCH** should be defined.

native-xtools-install

Installs the results to *${DESTDIR}/${NXTP}* where *NXTP* defaults to *nxb-bin*. **TARGET** and **TARGET_ARCH** must be defined.

packageworld Archive the results of **distributeworld**, placing the results in *DISTDIR*. This target is used while building a release; see release(7).

installworld Install everything built by a preceding **buildworld** step into the directory hierarchy pointed to by make(1) variable *DESTDIR*.

If installing onto an NFS file system and running make(1) with the **-j** option, make sure that rpc.lockd(8) is running on both client and server. See rc.conf(5) on how to make it start at boot time.

toolchain Create the build toolchain needed to build the rest of the system. For cross-architecture builds, this step creates a cross-toolchain.

universe For each architecture, execute a **buildworld** followed by a **buildkernel** for all kernels for that architecture, including *LINT*. This command takes a long time.

kernels Like **universe** with *WITHOUT_WORLDS* defined so only the kernels for each architecture are built.

worlds Like **universe** with *WITHOUT_KERNELS* defined so only the worlds for each architecture are built.

targets Print a list of supported *TARGET* / *TARGET_ARCH* pairs for world and kernel targets.

tinderbox Execute the same targets as **universe**. In addition print a summary of all failed targets at the end and exit with an error if there were any.

toolchains Create a build toolchain for each architecture supported by the build system.

xdev Builds and installs a cross-toolchain and sysroot for the given **TARGET** and **TARGET_ARCH**. The sysroot contains target library and headers. The target is an alias for **xdev-build** and **xdev-install**. The location of the files installed can be controlled with *DESTDIR*. The target location in *DESTDIR* is *${DESTDIR}/${XDTP}* where *XDTP* defaults to */usr/${XDDIR}* and *XDDIR* defaults to *${TARGET_ARCH}-freebsd*.

xdev-build Builds for the **xdev** target.

xdev-install Installs the files for the **xdev** target.

xdev-links Installs autoconf-style symlinks to *${DESTDIR}/usr/bin* pointing into the xdev toolchain in *${DESTDIR}/${XDTP}*.

Kernel specific build targets in */usr/src* are:

buildkernel Rebuild the kernel and the kernel modules. The object directory can be changed from the default */usr/obj* by setting the *MAKEOBJDIRPREFIX* make(1) variable.

installkernel Install the kernel and the kernel modules to directory *${DESTDIR}/boot/kernel*, renaming any pre-existing directory with this name to *kernel.old* if it contained the currently running kernel. The target directory under *${DESTDIR}* may be modified using the

INSTKERNNAME and *KODIR* make(1) variables.

distributekernel
Install the kernel to the directory *${DISTDIR}/kernel/boot/kernel*. This target is used while building a release; see release(7).

packages Create a pkg(7) repository containing packages that can be used to create or upgrade an installation of the base system. The output repository is placed in the object directory, under *repo/${PKG_ABI}* where *PKG_ABI* is the pkg(7) ABI for the build target, for example, */usr/obj/${SRCDIR}/repo/FreeBSD:15:amd64*.

packagekernel Archive the results of **distributekernel**, placing the results in *DISTDIR*. This target is used while building a release; see release(7).

kernel Equivalent to **buildkernel** followed by **installkernel**

kernel-toolchain
Rebuild the tools needed for kernel compilation. Use this if you did not do a **buildworld** first.

reinstallkernel Reinstall the kernel and the kernel modules, overwriting the contents of the target directory. As with the **installkernel** target, the target directory can be specified using the make(1) variable *INSTKERNNAME*.

Convenience targets for cleaning up the install destination directory denoted by variable *DESTDIR* include:

check-old Print a list of old files and directories in the system.

check-old-libs
Print a list of obsolete base system libraries.

delete-old Delete obsolete base system files and directories interactively. When -DBATCH_DELETE_OLD_FILES is specified at the command line, the delete operation will be non-interactive. The variables *DESTDIR*, *TARGET_ARCH* and *TARGET* should be set as with "make installworld".

delete-old-libs
Delete obsolete base system libraries interactively. This target should only be used if no third party software uses these libraries. When -DBATCH_DELETE_OLD_FILES is specified at the command line, the delete operation will be non-interactive. The variables

DESTDIR, *TARGET_ARCH* and *TARGET* should be set as with "make installworld".

ENVIRONMENT

Variables that influence all builds include:

DEBUG_FLAGS Defines a set of debugging flags that will be used to build all userland binaries under */usr/src*. When *DEBUG_FLAGS* is defined, the **install** and **installworld** targets install binaries from the current *MAKEOBJDIRPREFIX* without stripping, so that debugging information is retained in the installed binaries.

DESTDIR The directory hierarchy prefix where built objects will be installed. If not set, *DESTDIR* defaults to the empty string. If set, *DESTDIR* must specify an absolute path.

MAKEOBJDIRPREFIX

Defines the prefix for directory names in the tree of built objects. Defaults to */usr/obj* if not defined. This variable should only be set in the environment or */etc/src-env.conf* and not via */etc/make.conf* or */etc/src.conf* or the command line. *MAKEOBJDIRPREFIX* must specify an absolute path.

WITHOUT_WERROR If defined, compiler warnings will not cause the build to halt, even if the makefile says otherwise.

WITH_CTF If defined, the build process will run the DTrace CTF conversion tools on built objects.

Additionally, builds in */usr/src* are influenced by the following make(1) variables:

CROSS_TOOLCHAIN Requests use of an external toolchain to build either the world or kernel. This value of this variable can either be the full path to a file, or the base name of a file in *${LOCALBASE}/share/toolchains*. The file should be a make file which sets variables to request an external toolchain such as *XCC*.

External toolchains are available in ports for both LLVM and GCC/binutils. For external toolchains available in ports, *CROSS_TOOLCHAIN* should be set to the name of the package. LLVM toolchain packages use the name llvm<major version>. GCC toolchains provide separate packages for each architecture and use the name ${MACHINE_ARCH}-gcc<major version>.

KERNCONF Overrides which kernel to build and install for the various kernel make targets.

It defaults to **GENERIC**.

KERNBUILDDIR Overrides the default directory to get all the opt_*.h files for building a kernel module. Useful for stand-alone modules that depend on config(8) options. Automatically set for modules built with a kernel.

KERNCONFDIR Overrides the directory in which *KERNCONF* and any files included by *KERNCONF* should be found. Defaults to *sys/${ARCH}/conf*.

KERNFAST If set, the build target **buildkernel** defaults to setting *NO_KERNELCLEAN*, *NO_KERNELCONFIG*, and *NO_KERNELOBJ*. When set to a value other than **1** then *KERNCONF* is set to the value of *KERNFAST*.

LOCAL_DIRS If set, this variable supplies a list of additional directories relative to the root of the source tree to build as part of the **everything** target. The directories are built in parallel with each other, and with the base system directories. Insert a *.WAIT* directive at the beginning of the *LOCAL_DIRS* list to ensure all base system directories are built first. *.WAIT* may also be used as needed elsewhere within the list.

LOCAL_ITOOLS If set, this variable supplies a list of additional tools that are used by the **installworld** and **distributeworld** targets.

LOCAL_LIB_DIRS If set, this variable supplies a list of additional directories relative to the root of the source tree to build as part of the **libraries** target. The directories are built in parallel with each other, and with the base system libraries. Insert a *.WAIT* directive at the beginning of the *LOCAL_DIRS* list to ensure all base system libraries are built first. *.WAIT* may also be used as needed elsewhere within the list.

LOCAL_MTREE If set, this variable supplies a list of additional mtrees relative to the root of the source tree to use as part of the **hierarchy** target.

LOCAL_LEGACY_DIRS

If set, this variable supplies a list of additional directories relative to the root of the source tree to build as part of the **legacy** target.

LOCAL_BSTOOL_DIRS If set, this variable supplies a list of additional directories relative to the root of the source tree to build as part of the **bootstrap-tools** target.

LOCAL_TOOL_DIRS If set, this variable supplies a list of additional directories relative to the root of the source tree to build as part of the **build-tools** target.

LOCAL_XTOOL_DIRS If set, this variable supplies a list of additional directories relative to the root of the source tree to build as part of the **cross-tools** target.

PORTS_MODULES A list of ports with kernel modules that should be built and installed as part of the **buildkernel** and **installkernel** process.

make PORTS_MODULES=emulators/virtualbox-ose-kmod kernel

LOCAL_MODULES A list of external kernel modules that should be built and installed as part of the **buildkernel** and **installkernel** process. Defaults to the list of sub-directories of *LOCAL_MODULES_DIR*.

LOCAL_MODULES_DIR

The directory in which to search for the kernel modules specified by *LOCAL_MODULES*. Each kernel module should consist of a directory containing a makefile. Defaults to *${LOCALBASE}/sys/modules*.

SRCCONF Specify a file to override the default */etc/src.conf*. The src.conf file controls the components to build. See src.conf(5)

STRIPBIN Command to use at install time when stripping binaries. Be sure to add any additional tools required to run *STRIPBIN* to the *LOCAL_ITOOLS* make(1) variable before running the **distributeworld** or **installworld** targets. See install(1) for more details.

SUBDIR_OVERRIDE Override the default list of sub-directories and only build the sub-directory named in this variable. If combined with **buildworld** then all libraries and includes, and some of the build tools will still build as well. Specifying **-DNO_LIBS**, and **-DWORLDFAST** will only build the specified directory as was done historically. When combined with **buildworld** it is necessary to override *LOCAL_LIB_DIRS* with any custom directories containing libraries. This allows building a subset of the system in the same way as **buildworld** does using its sysroot handling. This variable can also be useful when debugging failed builds.

make some-target SUBDIR_OVERRIDE=foo/bar

SYSDIR Specify the location of the kernel source to override the default */usr/src/sys*. The kernel source is located in the *sys* subdirectory of the source tree checked out from the *src.git* repository.

TARGET The target hardware platform. This is analogous to the "**uname -m**" output. This is necessary to cross-build some target architectures. For example, cross-building for ARM64 machines requires *TARGET_ARCH*=aarch64 and *TARGET*=arm64. If not set, *TARGET* defaults to the current hardware platform, unless *TARGET_ARCH* is also set, in which case it defaults to the appropriate value for that architecture.

TARGET_ARCH The target machine processor architecture. This is analogous to the "**uname -p**" output. Set this to cross-build for a different architecture. If not set, *TARGET_ARCH* defaults to the current machine architecture, unless *TARGET* is also set, in which case it defaults to the appropriate value for that platform. Typically, one only needs to set *TARGET*.

Builds under directory */usr/src* are also influenced by defining one or more of the following symbols, using the **-D** option of make(1):

LOADER_DEFAULT_INTERP
Defines what interpreter the default loader program will have. Valid values include "4th", "lua", and "simp". This creates the default link for */boot/loader* to the loader with that interpreter. It also determines what interpreter is compiled into *userboot*.

NO_CLEANDIR If set, the build targets that clean parts of the object tree use the equivalent of "make clean" instead of "make cleandir".

NO_CLEAN If set, no object tree files are cleaned at all. This is the default when *WITH_META_MODE* is used with filemon(4) loaded. See src.conf(5) for more details. Setting *NO_CLEAN* implies *NO_KERNELCLEAN*, so when *NO_CLEAN* is set no kernel objects are cleaned either.

NO_CTF If set, the build process does not run the DTrace CTF conversion tools on built objects.

NO_SHARE If set, the build does not descend into the */usr/src/share* subdirectory (i.e., manual pages, locale data files, timezone data files and other */usr/src/share* files will not be rebuild from their sources).

NO_KERNELCLEAN If set, the build process does not run "make clean" as part of the **buildkernel** target.

NO_KERNELCONFIG If set, the build process does not run config(8) as part of the **buildkernel** target.

NO_KERNELOBJ If set, the build process does not run "make obj" as part of the **buildkernel** target.

NO_LIBS If set, the libraries phase will be skipped.

NO_OBJWALK If set, no object directories will be created. This should only be used if object directories were created in a previous build and no new directories are connected.

UNIVERSE_TOOLCHAIN Requests use of the toolchain built as part of the **universe** target as an external toolchain.

WORLDFAST If set, the build target **buildworld** defaults to setting *NO_CLEAN*, *NO_OBJWALK*, and will skip most bootstrap phases. It will only bootstrap libraries and build all of userland. This option should be used only when it is known that none of the bootstrap needs changed and that no new directories have been connected to the build.

Builds under directory */usr/doc* are influenced by the following make(1) variables:

DOC_LANG
If set, restricts the documentation build to the language subdirectories specified as its content. The default action is to build documentation for all languages.

Builds using the **universe** and related targets are influenced by the following make(1) variables:

JFLAG Pass the value of this variable to each make(1) invocation used to build worlds and kernels. This can be used to enable multiple jobs within a single architecture’s build while still building each architecture serially.

MAKE_JUST_KERNELS Only build kernels for each supported architecture.

MAKE_JUST_WORLDS Only build worlds for each supported architecture.

WITHOUT_WORLDS Only build kernels for each supported architecture.

WITHOUT_KERNELS Only build worlds for each supported architecture.

UNIVERSE_TARGET Execute the specified make(1) target for each supported architecture instead of the default action of building a world and one or more kernels. This variable implies *WITHOUT_KERNELS*.

USE_GCC_TOOLCHAINS

Use external GCC toolchains to build the requested targets. If the required toolchain package for a supported architecture is not installed, the build for that architecture is skipped.

A specific version of GCC can be used by setting the value of this variable to the desired version (for example, "gcc14"); otherwise a default version of GCC is used.

TARGETS Only build the listed targets instead of each supported architecture.

EXTRA_TARGETS In addition to the supported architectures, build the semi-supported architectures. A semi-supported architecture has build support in the FreeBSD tree, but receives significantly less testing and is generally for fringe uses that do not have a wide appeal.

FILES

/usr/doc/Makefile
/usr/doc/share/mk/doc.project.mk
/usr/ports/Mk/bsd.port.mk
/usr/ports/Mk/bsd.sites.mk
/usr/src/Makefile
/usr/src/Makefile.inc1 make(1) infrastructure for each tree
/usr/ports/UPDATING
/usr/src/UPDATING manual intervention required for updating each tree
/usr/share/examples/etc/make.conf
example make.conf(5)
/etc/src.conf src build configuration, see src.conf(5)

EXAMPLES

For an "approved" method of updating your system from the latest sources, please see the *COMMON ITEMS* section in *src/UPDATING*.

Build and upgrade system in place

If using installed drivers such as graphics or guest drivers, check out the ports(7) tree, and specify the drivers in src.conf(5) so they are built and installed automatically after the kernel:

```
git clone https://git.FreeBSD.org/ports.git /usr/ports
cat >> EOF >> /etc/src.conf
PORTS_MODULES+=graphics/drm-kmod emulators/virtualbox-ose-kmod
EOF
```

Check out the CURRENT branch, build it, and install, overwriting the current system:

```
git clone https://git.FreeBSD.org/src.git /usr/src
cd /usr/src
make -sj8 buildworld kernel
shutdown -r now
```

After reboot, install userspace, merge configurations, and delete old files:

```
cd src
etcupdate -p
make -j8 installworld
etcupdate -B
make delete-old
reboot
```

Build and upgrade a custom kernel in place

Create a custom kernel configuration, *MYKERNEL*, by including an existing configuration and using **device**/**nodevice** and **options**/**nooption** to select and configure components:

```
cd src
cat >> EOF > sys/amd64/conf/MYKERNEL
include GENERIC
ident MYKERNEL
nodevice sound
EOF
```

After creating the new kernel configuration, build it, and install, moving the old kernel to */boot/kernel.old/*:

```
make -j8 kernel KERNCONF=MYKERNEL
reboot
```

Build and upgrade a single piece of userspace
Rebuild and reinstall a single piece of userspace, in this case ls(1):

```
cd src/bin/ls
make clean all install
```

Build and upgrade a loadable kernel module
Rebuild and reinstall a single loadable kernel module, in this case sound(4):

```
cd src/sys/modules/sound
make all install clean cleandepend KMODDIR=/boot/kernel
```

Quickly rebuild a kernel in place
Quickly rebuild and reinstall the kernel, only recompiling the files changed since last build; note that this will only work if the full kernel build has been completed in the past, not on a fresh source tree:

```
cd src
make -sj8 kernel KERNFAST=1
```

Cross-compiling for different architectures
To rebuild parts of FreeBSD for another CPU architecture, first prepare your source tree by building the cross-toolchain:

```
cd src
make -sj8 toolchain TARGET_ARCH=aarch64
```

The following sequence of commands can be used to cross-build the system for the arm64 (aarch64) architecture on a different host architecture, such as amd64:

```
cd /usr/src
make TARGET=arm64 buildworld buildkernel
make TARGET=arm64 DESTDIR=/clients/arm64 installworld installkernel
```

Afterwards, to build and install a single piece of userspace, use:

```
cd src/bin/ls
make buildenv TARGET_ARCH=aarch64
make clean all install DESTDIR=/clients/arm
```

Likewise, to quickly rebuild and reinstall the kernel, use:

```
cd src
make buildenv TARGET_ARCH=aarch64
make -sj8 kernel KERNFAST=1 DESTDIR=/clients/arm
```

SEE ALSO

cc(1), install(1), make(1), make.conf(5), src.conf(5), arch(7), development(7), pkg(7), ports(7), release(7), tests(7), config(8), etcupdate(8), reboot(8), shutdown(8)

HISTORY

The **build** manpage first appeared in FreeBSD 4.3.

AUTHORS

Mike W. Meyer <*mwm@mired.org*>

CAVEATS

Environment poisioning can cause obscure build problems, try prefixing make(1) commands with ‘env -i’

When doing a major release upgrade, booting into single user mode for installworld is required.

BUGS

Documentation on building the system is spread out over a lot of places.