The pkgsrc guideDocumentation on the NetBSD packages systemAlistairCrooksagc@NetBSD.orgHubertFeyrerhubertf@NetBSD.org The pkgsrc Developers 1994-2004The NetBSD Foundation, Inc$NetBSD: pkgsrc.xml,v 1.2 2004/10/21 15:07:47 grant Exp $Information about using the NetBSD package system (pkgsrc) from both a user view for installing packages as well as from a pkgsrc developers' view for creating new packages.IntroductionIntroduction There is a lot of software freely available for Unix based systems, which usually runs on NetBSD and other Unix-flavoured systems, too, sometimes with some modifications. The NetBSD Packages Collection (pkgsrc) incorporates any such changes necessary to make that software run, and makes the installation (and de-installation) of the software package easy by means of a single command. Once the software has been built, it is manipulated with the pkg_* tools so that installation and de-installation, printing of an inventory of all installed packages and retrieval of one-line comments or more verbose descriptions are all simple.pkgsrc currently contains several thousand packages, including:www/apache - The Apache web serverwww/mozilla - The Mozilla web browsermeta-pkgs/gnome - The GNOME Desktop Environmentmeta-pkgs/kde3 - The K Desktop Environment...just to name a few.pkgsrc has built-in support for handling varying dependencies, such as pthreads and X11, and extended features such as IPv6 support on a range of platforms.pkgsrc was derived from FreeBSD's ports system, and initially developed for NetBSD only. Since then, pkgsrc has grown a lot, and now supports the following platforms:Darwin (Mac OS X)DragonFlyBSDFreeBSDMicrosoft Windows, via InterixIRIXLinuxNetBSD (of course)OpenBSDSolarisOverviewThis document is divided into two parts. The first, , describes how one can use one of the packages in the Package Collection, either by installing a precompiled binary package, or by building one's own copy using the NetBSD package system. The second part, , explains how to prepare a package so it can be easily built by other NetBSD users without knowing about the package's building details.This document is available in various formats:HTMLPDFPSTXTTerminologyThere has been a lot of talk about ports, packages, etc. so far. Here is a description of all the terminology used within this document.PackageA set of files and building instructions that describe what's necessary to build a certain piece of software using pkgsrc. Packages are traditionally stored under /usr/pkgsrc.The NetBSD package system This is the former name of pkgsrc. It is part of the NetBSD operating system and can be bootstrap to run on non-NetBSD operating systems as well. It handles building (compiling), installing, and removing of packages. DistfileThis term describes the file or files that are provided by the author of the piece of software to distribute his work. All the changes necessary to build on NetBSD are reflected in the corresponding package. Usually the distfile is in the form of a compressed tar-archive, but other types are possible, too. Distfiles are usually stored below /usr/pkgsrc/distfiles.PortThis is the term used by FreeBSD and OpenBSD people for what we call a package. In NetBSD terminology, port refers to a different architecture.Precompiled/binary packageA set of binaries built with pkgsrc from a distfile and stuffed together in a single .tgz file so it can be installed on machines of the same machine architecture without the need to recompile. Packages are usually generated in /usr/pkgsrc/packages; there is also an archive on ftp.NetBSD.org.Sometimes, this is referred to by the term package too, especially in the context of precompiled packages.ProgramThe piece of software to be installed which will be constructed from all the files in the Distfile by the actions defined in the corresponding package.TypographyWhen giving examples for commands, shell prompts are used to show if the command should/can be issued as root, or if normal user privileges are sufficient. We use a # for root's shell prompt, and a $ for users' shell prompt, assuming they use the C-shell or tcsh.The pkgsrc user's guideWhere to get pkgsrcThere are three ways to get pkgsrc. Either as a tar file, via SUP, or via CVS. All three ways are described here.As tar fileTo get pkgsrc going, you need to get the pkgsrc.tar.gz file from ftp.NetBSD.org and unpack it into /usr/pkgsrc.Via SUPAs an alternative to the tar file, you can get pkgsrc via the Software Update Protocol, SUP. To do so, make sure your supfile has a linerelease=pkgsrcin it, see the examples in /usr/share/examples/supfiles, and that the /usr/pkgsrc directory exists. Then, simply run sup -v /path/to/your/supfile.Via CVSTo get pkgsrc via CVS, make sure you have cvs installed. If not present on your system, it can be found as precompiled binary on ftp.NetBSD.org. To do an initial (full) checkout of pkgsrc, do the following steps:% setenv CVSROOT anoncvs@anoncvs.NetBSD.org:/cvsroot % setenv CVS_RSH ssh % cd /usr % cvs checkout -P pkgsrcThis will create the pkgsrc directory in your /usr, and all the package source will be stored under /usr/pkgsrc. To update pkgsrc after the initial checkout, make sure you have CVS_RSH set as above, then do:% cd /usr/pkgsrc % cvs -q update -dPPlease also note that it is possible to have multiple copies of the pkgsrc hierarchy in use at any one time - all work is done relatively within the pkgsrc tree.Using pkgsrc on systems other than NetBSDBootstrapping pkgsrcFor Operating Systems other than NetBSD, we provide a bootstrap kit to build the required tools to use pkgsrc on your platform. Besides support for native NetBSD, pkgsrc and the bootstrap kit have support for the following operating systems:Darwin (Mac OS X)FreeBSDInterix (Windows 2000, XP, 2003)IRIXLinuxOpenBSDSolarisSupport for other platforms is under development.Installing the bootstrap kit should be as simple as: # env CVS_RSH=ssh cvs -d anoncvs@anoncvs.NetBSD.org:/cvsroot checkout pkgsrc # cd pkgsrc/bootstrap # ./bootstrapSee for other ways to get pkgsrc before bootstrapping. The given bootstrap command will use the defaults of /usr/pkg for the prefix where programs will be installed in, and /var/db/pkg for the package database directory where pkgsrc will do it's internal bookkeeping. However, these can also be set using command-line parameters.Binary packages for the pkgsrc tools and an initial set of packages is available for supported platforms. An up-to-date list of these can be found on www.pkgsrc.org.Platform specific notesHere are some platform-specific notes you should be aware of.Darwin (Mac OS X)Darwin 5.x and 6.x are supported. There are two methods of using pkgsrc on Mac OS X, by using a disk image, or a UFS partition.Before you start, you will need to download and install the Mac OS X Developer Tools from Apple's Developer Connection. See http://developer.apple.com/macosx/ for details. Also, make sure you install X11 for Mac OS X and the X11 SDK from http://www.apple.com/macosx/x11/download/ if you intend to build packages that use the X11 Window System.If you already have a UFS partition, or have a spare partition that you can format as UFS, it is recommended to use that instead of the disk image. It'll be somewhat faster and will mount automatically at boot time, where you must manually mount a disk image.You cannot use a HFS+ file system for pkgsrc, because pkgsrc currently requires the filesystem to be case-sensitive, and HFS+ is not.Using a disk imageCreate the disk image:# cd pkgsrc/bootstrap # ./ufsdiskimage create ~/Documents/NetBSD 512 # megabytes - season to taste # ./ufsdiskimage mount ~/Documents/NetBSD # sudo chown 'id -u':'id -g' /Volumes/NetBSDThat's it!Using a UFS partitionBy default, /usr will be on your root file system, normally HFS+. It is possible to use the default prefix of /usr/pkg by symlinking /usr/pkg to a directory on a UFS file system. Obviously, another symlink is required if you want to place the package database directory outside the prefix. e.g.# ./bootstrap --pkgdbdir=/usr/pkg/pkgdb --pkgsrcdir=/Volumes/ufs/pkgsrcIf you created your partitions at the time of installing Mac OS X and formatted the target partition as UFS, it should automatically mount on /Volumes/<volume name> when the machine boots. If you are (re)formatting a partition as UFS, you need to ensure that the partition map correctly reflects Apple_UFS and not Apple_HFS.The problem is that none of the disk tools will let you touch a disk that is booted from. You can unmount the partition, but even if you newfs it, the partition type will be incorrect and the automounter won't mount it. It can be mounted manually, but it won't appear in Finder.You'll need to boot off of the OS X Installation (User) CD. When the Installation program starts, go up to the menu and select Disk Utility. Now, you will be able to select the partition you want to be UFS, and Format it Apple UFS. Quit the Disk Utility, quit the installer which will reboot your machine. The new UFS file system will appear in Finder.Be aware that the permissions on the new file system will be writable by root only.This note is as of 10.2 (Jaguar) and applies to earlier versions. Hopefully Apple will fix Disk Utility in 10.3 (Panther).FreeBSD FreeBSD 4.7 and 5.0 have been tested and are supported, other versions may work.Care should be taken so that the tools that this kit installs do not conflict with the FreeBSD userland tools. There are several steps:FreeBSD stores its ports pkg database in /var/db/pkg. It is therefore recommended that you choose a different location (e.g. /usr/pkgdb) by using the --pkgdbdir option to the bootstrap script.If you do not intend to use the FreeBSD ports tools, it's probably a good idea to move them out of the way to avoid confusion, e.g.# cd /usr/sbin # mv pkg_add pkg_add.orig # mv pkg_create pkg_create.orig # mv pkg_delete pkg_delete.orig # mv pkg_info pkg_info.origAn example /etc/mk.conf file will be placed in /etc/mk.conf.example file when you use the bootstrap script.InterixInterix is a POSIX compatible subsystem for the Windows NT kernel, providing a Unix-like environment with a tighter kernel integration than available with Cygwin. It is part of the Windows Services for Unix package, available for free for any licensed copy of Windows 2000, XP, or 2003. SFU can be downloaded from http://www.microsoft.com/windows/sfu/.Services for Unix 3.5, current as of this writing, has been tested. 3.0 or 3.1 may work, but are not officially supported. (The main difference in 3.0/3.1 is lack of pthreads.)When installing Interix/SFUAt an absolute minimum, the following packages must be installed from the Windows Services for Unix 3.5 distribution in order to use pkgsrc:Utilities -> Base UtilitiesInterix GNU Components -> (all)Remote ConnectivityInterix SDKWhen using pkgsrc on Interix, DO NOT install the Utilities subcomponent "UNIX Perl". That is Perl 5.6 without shared module support, installed to /usr/local, and will only cause confusion. Instead, install Perl 5.8 from pkgsrc (or from a binary package).The Remote Connectivity subcomponent "Windows Remote Shell Service" does not need to be installed, but Remote Connectivity itself should be installed in order to have a working inetd.Finally, during installation you may be asked whether to enable setuid behavior for Interix programs, and whether to make pathnames default to case-sensitive. Setuid should be enabled, and case-sensitivity MUST be enabled. (Without case-sensitivity, a large number of packages including perl will not build.)What to do if Interix/SFU is already installedIf SFU is already installed and you wish to alter these settings to work with pkgsrc, note the following things.To uninstall UNIX Perl, use Add/Remove Programs, select Microsoft Windows Services for UNIX, then click Change. In the installer, choose Add or Remove, then uncheck Utilities->UNIX Perl.To enable case-sensitivity for the filesystem, run REGEDIT.EXE, and change the following registry key:HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Session Manager\kernelSet the DWORD value "obcaseinsensitive" to 0; then reboot.To enable setuid binaries (optional), run REGEDIT.EXE, and change the following registry key:HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Services for UNIXSet the DWORD value "EnableSetuidBinaries" to 1; then reboot.Important notes for using pkgsrcThe package imanager (either the pkgsrc "su" user, or the user running "pkg_add") must be a member of the local Administrators group. Such a user must also be used to run the bootstrap. This is slightly relaxed from the normal pkgsrc requirement of "root".The package manager should use a umask of 002. "make install" will automatically complain if this is not the case. This ensures that directories written in /var/db/pkg are Administrators-group writeable.The popular Interix binary packages from http://www.interopsystems.com/ use an older version of pkgsrc's pkg_* tools. Ideally, these should NOT be used in conjunction with pkgsrc. If you choose to use them at the same time as the pkgsrc packages, ensure that you use the proper pkg_* tools for each type of binary package.IRIXYou will need a working C compiler, either gcc or SGI's MIPS and MIPSpro compiler (cc/c89). Please set the CC environment variable according to your preference. If you do not have a license for the MIPSpro compiler suite, you can download a gcc tardist file from http://freeware.sgi.com/.Please note that you will need IRIX 6.5.17 or higher, as this is the earliest version of IRIX providing support for if_indextoname(3), if_nametoindex(3), etc.At this point in time, pkgsrc only supports one ABI. That is, you can not switch between the old 32-bit ABI, the new 32-bit ABI and the 64-bit ABI. If you start out using "abi=n32", that's what all your packages will be built with.Therefore, please make sure that you have no conflicting CFLAGS in your environment or the /etc/mk.conf. Particularly, make sure that you do not try to link n32 object files with lib64 or vice versa. Check your /etc/compiler.defaults!If you have the actual pkgsrc tree mounted via NFS from a different host, please make sure to set WRKOBJDIR to a local directory, as it appears that IRIX linker occasionally runs into issues when trying to link over a network mounted filesystem.The bootstrapping process should set all the right options for programs such as imake(1), but you may want to set some options depending on your local setup. Please see pkgsrc/mk/defaults/mk.conf and, of course, your compilers man pages for details.OpenBSDOpenBSD 3.0 and 3.2 are tested and supported.Care should be taken so that the tools that this kit installs do not conflict with the OpenBSD userland tools. There are several steps:OpenBSD stores its ports pkg database in /var/db/pkg. It is therefore recommended that you choose a different location (e.g. /usr/pkgdb) by using the --pkgdbdir option to the bootstrap script.If you do not intend to use the OpenBSD ports tools, it's probably a good idea to move them out of the way to avoid confusion, e.g.# cd /usr/sbin # mv pkg_add pkg_add.orig # mv pkg_create pkg_create.orig # mv pkg_delete pkg_delete.orig # mv pkg_info pkg_info.origAn example /etc/mk.conf file will be placed in /etc/mk.conf.example file when you use the bootstrap script. OpenBSD's make program uses /etc/mk.conf as well. You can work around this by enclosing all the pkgsrc specific parts of the file with:.ifdef BSD_PKG_MK # pkgsrc stuff, e.g. insert defaults/mk.conf or similar here .else # OpenBSD stuff .endifSolarisSolaris 2.6 through 9 are supported on both x86 and sparc. You will need a working C compiler. Both gcc 2.95.3 and Sun WorkShop 5 have been tested.The following packages are required on Solaris 8 for the bootstrap process and to build packages.SUNWsprotSUNWarcSUNWbtoolSUNWtooSUNWlibmPlease note the use of GNU binutils on Solaris is not supported.If you are using gccIt makes life much simpler if you only use the same gcc consistently for building all packages.It is recommended that an external gcc be used only for bootstrapping, then either build gcc from lang/gcc or install a binary gcc package, then remove gcc used during bootstrapping.Binary packages of gcc can be found through http://www.sun.com/bigadmin/common/freewareSearch.html.If you are using Sun WorkShopYou will need at least the following packages installed (from WorkShop 5.0)SPROcc - Sun WorkShop Compiler C 5.0SPROcpl - Sun WorkShop Compiler C++ 5.0SPROild - Sun WorkShop Incremental LinkerSPROlang - Sun WorkShop Compilers common componentsYou should set CC, CXX and optionally, CPP in /etc/mk.conf, eg.CC= cc CXX= CC CPP= /usr/ccs/lib/cppYou may also want to build 64-bit binaries, eg.CFLAGS= -xtarget=ultra -xarch=v9Whichever compiler you use, please ensure the compiler tools and your $prefix are in your PATH. This includes /usr/ccs/{bin,lib} and eg. /usr/pkg/{bin,sbin}.Using pkgsrcWorking with binary packages This section describes how to find, retrieve and install a precompiled binary package that someone else already prepared for your type of machine. Where to get binary packages Precompiled packages are stored on ftp.NetBSD.org and its mirrors in the directory /pub/NetBSD/packages for anonymous FTP access. Please pick the right subdirectory there as indicated by uname -p. In that directory, there is a subdirectory for each category plus a subdirectory All which includes the actual binaries in .tgz files. The category subdirectories use symbolic links to those files (this is the same directory layout as in /usr/pkgsrc/packages). This same directory layout applies for CDROM distributions, only that the directory may be rooted somewhere else, probably somewhere below /cdrom. Please consult your CDROMs documentation for the exact location. How to use binary packages If you have the files on a CDROM or downloaded them to your hard disk, youcan install them with the following command (be sure tosu to root first):# pkg_add /path/to/package.tgz If you have FTP access and you don't want to download the packages via FTP prior to installation, you can do this automatically by giving pkg_add an FTP URL: # pkg_add ftp://ftp.NetBSD.org/pub/NetBSD/packages/<OSvers>/<arch>/All/package.tgzIf there is any doubt, the uname utility can be used to determine the <OSvers>, and <arch> by running uname -rp. Also note that any prerequisite packages needed to run the package in question will be installed, too, assuming they are present where you install from. After you've installed packages, be sure to have /usr/pkg/bin in your PATH so you can actually start the just installed program. A word of warning Please pay very careful attention to the warnings expressed in the pkg_add1 manual page about the inherent dangers of installing binary packages which you did not create yourself, and the security holes that can be introduced onto your system by indiscriminate adding of such files. Building packages from source This assumes that the package is already in pkgsrc. If it is not, see . Requirements To build packages from source on a NetBSD system the comp and the text distribution sets must be installed. If you want to build X11 related packages the xbase and xcomp distribution sets are required, too.Fetching distfilesThe distfile (i.e. the unmodified source) must exist on your system for the packages system to be able to build it. If it does not exist, pkgsrc will use ftp1 to fetch it automatically.You can overwrite some of the major distribution sites to fit to sites that are close to your own. Have a look at pkgsrc/mk/defaults/mk.conf to find some examples - in particular, look for the MASTER_SORT, MASTER_SORT_REGEX and INET_COUNTRY definitions. This may save some of your bandwidth and time.You can change these settings either in your shell's environment, or, if you want to keep the settings, by editing the /etc/mk.conf file, and adding the definitions there.If you don't have a permanent Internet connection and you want to know which files to download, make fetch-list will tell you what you'll need. Put these distfiles into /usr/pkgsrc/distfiles.How to build and installAssuming that the distfile has been fetched (see previous section), become root and change into the relevant directory and running make. For example, type % cd misc/figlet % makeat the shell prompt to build the various components of the package, and# make installto install the various components into the correct places on your system. Installing the package on your system requires you to be root. However, pkgsrc has a just-in-time-su feature, which allows you to only become root for the actual installation stepTaking the figlet utility as an example, we can install it on our system by building as shown in .The program is installed under the default root of the packages tree - /usr/pkg. Should this not conform to your tastes, set the LOCALBASE variable in your environment, and it will use that value as the root of your packages tree. So, to use /usr/local, set LOCALBASE=/usr/local in your environment. Please note that you should use a directory which is dedicated to packages and not shared with other programs (ie, do not try and use LOCALBASE=/usr). Also, you should not try to add any of your own files or directories (such as src/, obj/, or pkgsrc/) below the LOCALBASE tree. This is to prevent possible conflicts between programs and other files installed by the package system and whatever else may have been installed there.Some packages look in /etc/mk.conf to alter some configuration options at build time. Have a look at pkgsrc/mk/defaults/mk.conf to get an overview of what will be set there by default. Environment variables such as LOCALBASE can be set in /etc/mk.conf to save having to remember to set them each time you want to use pkgsrc.Occasionally, people want to look under the covers to see what is going on when a package is building or being installed. This may be for debugging purposes, or out of simple curiosity. A number of utility values have been added to help with this.If you invoke the make1 command with PKG_DEBUG_LEVEL=2, then a huge amount of information will be displayed. For example,make patch PKG_DEBUG_LEVEL=2will show all the commands that are invoked, up to and including the patch stage.If you want to know the value of a certain make(1) definition, then the VARNAME definition should be used, in conjunction with the show-var target. e.g. to show the expansion of the make1 variable DISTFILES:% make show-var VARNAME=LOCALBASE /usr/pkg % If you want to install a binary package that you've either created yourself (see next section), that you put into pkgsrc/packages manually or that is located on a remote FTP server, you can use the "bin-install" target. This target will install a binary package - if available - via pkg_add1, else do a make package. The list of remote FTP sites searched is kept in the variable BINPKG_SITES, which defaults to ftp.NetBSD.org. Any flags that should be added to pkg_add1 can be put into BIN_INSTALL_FLAGS. See pkgsrc/mk/defaults/mk.conf for more details.A final word of warning: If you setup a system that has a non-standard setting for LOCALBASE, be sure to set that before any packages are installed, as you can not use several directories for the same purpose. Doing so will result in pkgsrc not being able to properly detect your installed packages, and fail miserably. Note also that precompiled binary packages are usually built with the default LOCALBASE of /usr/pkg, and that you should not install any if you use a non-standard LOCALBASE.Selecting the compilerBy default, pkgsrc will use GCC to build packages. This may be overridden by setting the following variables in /etc/mk.conf: PKGSRC_COMPILER: This is a list of values specifying the chain of compilers to invoke when building packages. Valid values are: distcc: distributed C/C++ (chainable)ccache: compiler cache (chainable)gcc: GNU

C/C++ Compilermipspro: Silicon Graphics, Inc. MIPSpro (n32/n64)mipspro: Silicon Graphics, Inc. MIPSpro (o32)sunpro: Microsystems, Inc. WorkShip/Forte/Sun ONE Studio The default is gcc. You can use ccache and/or distcc with an appropriate PKGSRC_COMPILER setting, e.g. ccache gcc. This variable should always be terminated with a value for a real compiler. GCC_REQD: This specifies the minimum version of GCC to use when building packages. If the system GCC doesn't satisfy this requirement, then pkgsrc will build and install one of the GCC packages to use instead. Creating binary packagesBuilding a single binary packageOnce you have built and installed a package, you can create a binary package which can be installed on another system with pkg_add1 This saves having to build the same package on a group of hosts and wasting CPU time. It also provides a simple means for others to install your package, should you distribute it.To create a binary package, change into the appropriate directory in pkgsrc, and run make package:# cd misc/figlet # make packageThis will build and install your package (if not already done), and then build a binary package from what was installed. You can then use the pkg_* tools to manipulate it. Binary packages are created by default in /usr/pkgsrc/packages, in the form of a gzipped tar file. See for a continuation of the above misc/figlet example.See for information on how to submit such a binary package.Settings for creation of binary packagesSee .Doing a bulk build of all packagesIf you want to get a full set of precompiled binary packages, this section describes how to get them. Beware that the bulk build will remove all currently installed packages from your system! Having a FTP server configured either on the machine doing the bulk builds or on a nearby NFS server can help to make the packages available to everyone. See ftpd8 for more information. If you use a remote NFS server's storage, be sure to not actually compile on NFS storage, as this slows things down a lot.Configuration/etc/mk.confYou may want to set things in /etc/mk.conf. Look at pkgsrc/mk/defaults/mk.conf for details of the default settings. You will want to ensure that ACCEPTABLE_LICENSES meet your local policy. As used in this example, _ACCEPTABLE=yes accepts all licenses.PACKAGES?= ${_PKGSRCDIR}/packages/${MACHINE_ARCH} WRKOBJDIR?= /usr/tmp/pkgsrc # build here instead of in pkgsrc BSDSRCDIR= /usr/src BSDXSRCDIR= /usr/xsrc # for x11/xservers OBJHOSTNAME?= yes # use work.‘hostname‘ FAILOVER_FETCH= yes # insist on the correct checksum PKG_DEVELOPER?= yes _ACCEPTABLE= yesbuild.confIn pkgsrc/mk/bulk, copy build.conf-example to build.conf and edit it, following the comments in that file. This is the config file that determines where log files are generated after the build, where to mail the build report to, where your pkgsrc tree is located and which user to su8 to to do a cvs update.pre-build.localIt is possible to configure the bulk build to perform certain site specific tasks at the end of the pre-build stage. If the file pre-build.local exists in /usr/pkgsrc/mk/bulk it will be executed (as a sh(1) script) at the end of the usual pre-build stage. An example use of pre-build.local is to have the line:# echo "I do not have enough disk space to build this pig." \ > pkgsrc/games/crafty-book-enormous/$BROKENFto prevent the system from trying to build a particular package which requires nearly 3 GB of disk space.Other environmental considerationsAs /usr/pkg will be completely deleted at the start of bulk builds, make sure your login shell is placed somewhere else. Either drop it into /usr/local/bin (and adjust your login shell in the passwd file), or (re-)install it via pkg_add1 from /etc/rc.local, so you can login after a reboot (remember that your current process won't die if the package is removed, you just can't start any new instances of the shell any more). Also, if you use NetBSD earlier than 1.5, or you still want to use the pkgsrc version of ssh for some reason, be sure to install ssh before starting it from rc.local:(cd /usr/pkgsrc/security/ssh ; make bulk-install) if [-f /usr/pkg/etc/rc.d/sshd]; then /usr/pkg/etc/rc.d/sshd fiNot doing so will result in you being not able to log in via ssh after the bulk build is finished or if the machine gets rebooted or crashes. You have been warned! :)OperationMake sure you don't need any of the packages still installed.During the bulk build, all packages will be removed!Be sure to remove all other things that might interfere with builds, like some libs installed in /usr/local, etc. then become root and type:# cd /usr/pkgsrc # sh mk/bulk/buildIf for some reason your last build didn't complete (power failure, system panic, ...), you can continue it by running:# sh mk/bulk/build restartAt the end of the bulk build, you will get a summary via mail, and find build logs in the directory specified by FTP in the build.conf file.What it doesThe bulk builds consist of three steps:1. pre-buildThe script updates your pkgsrc tree via (anon)cvs, then cleans out any broken distfiles, and removes all packages installed.2. the bulk buildThis is basically make bulk-package with an optimised order in which packages will be built. Packages that don't require other packages will be built first, and packages with many dependencies will be built later.3. post-buildGenerates a report that's placed in the directory specified in the build.conf file named broken.html, a short version of that report will also be mailed to the build's admin.During the build, a list of broken packages will be compiled in /usr/pkgsrc/.broken (or .../.broken.${MACHINE} if OBJMACHINE is set), individual build logs of broken builds can be found in the package's directory. These files are used by the bulk-targets to mark broken builds to not waste time trying to rebuild them, and they can be used to debug these broken package builds later.Disk space requirementsCurrently, roughly the following requirements are valid for NetBSD 2.0/i386:10 GB - distfiles (NFS ok)8 GB - full set of all binaries (NFS ok)5 GB - temp space for compiling (local disk recommended)Note that all pkgs will be de-installed as soon as they are turned into a binary package, and that sources are removed, so there is no excessively huge demand to disk space. Afterwards, if the package is needed again, it will be installed via pkg_add1 instead of building again, so there are no cycles wasted by recompiling.Setting up a sandbox for chroot'ed buildsIf you don't want all the pkgs nuked from a machine (rendering it useless for anything but pkg compiling), there is the possibility of doing the pkg bulk build inside a chroot environment.The first step to do so is setting up a chroot sandbox, e.g. /usr/sandbox. After extracting all the sets from a NetBSD installation or doing a make distribution DESTDIR=/usr/sandbox in /usr/src/etc, be sure the following items are present and properly configured:Kernel# cp /netbsd /usr/sandbox/dev/*# cd /usr/sandbox/dev ; sh MAKEDEV all/etc/resolv.conf (for security/smtpd and mail):# cp /etc/resolv.conf /usr/sandbox/etcWorking(!) mail config (hostname, sendmail.cf):# cp /etc/mail/sendmail.cf /usr/sandbox/etc/mail/etc/localtime (for security/smtpd):# ln -sf /usr/share/zoneinfo/UTC /usr/sandbox/etc/localtime/usr/src (system sources, for sysutils/aperture, net/ppp-mppe):# ln -s ../disk1/cvs . # ln -s cvs/src-1.6 srcCreate /var/db/pkg (not part of default install):# mkdir /usr/sandbox/var/db/pkgCreate /usr/pkg (not part of default install):# mkdir /usr/sandbox/usr/pkgCheckout pkgsrc via cvs into /usr/sandbox/usr/pkgsrc:# cd /usr/sandbox/usr # cvs -d anoncvs@anoncvs.NetBSD.org:/cvsroot checkout -d -P pkgsrcDo not mount/link this to the copy of your pkgsrc tree you do development in, as this will likely cause problems! Make /usr/sandbox/usr/pkgsrc/packages and .../distfiles point somewhere appropriate. NFS- and/or nullfs-mounts may come in handy! Edit /etc/mk.conf, see .Adjust mk/bulk/build.conf to suit your needs.If you have set CVS_USER in build.conf, make sure that account exists and can do a cvs ${CVS_FLAGS} update properly!When the chroot sandbox is setup, you can start the build with the following steps:# cd /usr/sandbox/usr/pkgsrc # sh mk/bulk/do-sandbox-buildThis will just jump inside the sandbox and start building. At the end of the build, mail will be sent with the results of the build. Created binary pkgs will be in /usr/sandbox/usr/pkgsrc/packages (wherever that points/mounts to/from).Building a partial set of packages In addition to building a complete set of all packages in pkgsrc, the pkgsrc/mk/bulk/build script may be used to build a subset of the packages contained in pkgsrc. By setting defining SPECIFIC_PKGS in /etc/mk.conf, the variablesSITE_SPECIFIC_PKGSHOST_SPECIFIC_PKGSGROUP_SPECIFIC_PKGSUSER_SPECIFIC_PKGS will define the set of packages which should be built. The bulk build code will also include any packages which are needed as dependencies for the explicitly listed packages. One use of this is to do a bulk build with SPECIFIC_PKGS in a chroot sandbox periodically to have a complete set of the binary packages needed for your site available without the overhead of building extra packages that are not needed. Creating a multiple CD-ROM packages collectionAfter your pkgsrc bulk-build has completed, you may wish to create a CD-ROM set of the resulting binary packages to assist in installing packages on other machines. The pkgtools/cdpack package provides a simple tool for creating the ISO 9660 images. cdpack arranges the packages on the CD-ROMs in a way that keeps all the dependencies for given package on the same CD as that package.Example of cdpackComplete documentation for cdpack is found in the cdpack(1) manpage. The following short example assumes that the binary packages are left in /usr/pkgsrc/packages/All and that sufficient disk space exists in /u2 to hold the ISO 9660 images.# mkdir /u2/images # pkg_add /usr/pkgsrc/packages/All/cdpack # cdpack /usr/pkgsrc/packages/All /u2/imagesIf you wish to include a common set of files (COPYRIGHT, README, etc.) on each CD in the collection, then you need to create a directory which contains these files. e.g.# mkdir /tmp/common # echo "This is a README" > /tmp/common/README # echo "Another file" > /tmp/common/COPYING # mkdir /tmp/common/bin # echo "#!/bin/sh" > /tmp/common/bin/myscript # echo "echo Hello world" >> /tmp/common/bin/myscript # chmod 755 /tmp/common/bin/myscriptNow create the images:# cdpack -x /tmp/common /usr/pkgsrc/packages/All /u2/imagesEach image will contain README, COPYING, and bin/myscript in their root directories.Frequently Asked Questions This section contains hints, tips & tricks on special things in pkgsrc that we didn't find a better place for in the previous chapters, and it contains items for both pkgsrc users and developers. Is there a mailing list for pkg-related discussion? Yes, tech-pkg@NetBSD.org is the list for discussing package related issues. To subscribe do: % echo subscribe tech-pkg | mail majordomo@NetBSD.org An archive of the list is available at http://mail-index.NetBSD.org/tech-pkg/. Where's the pkgviews documentation? Pkgviews is tightly integrated with buildlink. You can find a pkgviews User's guide in pkgsrc/mk/buildlink3/PKGVIEWS_UG. Utilities for package management (pkgtools) The pkgsrc/pkgtools directory pkgtools contains a number of useful utilities for both users and developers of pkgsrc. This section attempts only to make the reader aware of the utilities and when they might be useful, and not to duplicate the documentation that comes with each package. Utilities used by pkgsrc (automatically installed when needed): pkgtools/x11-links: symlinks for use by buildlink OS tool augmentation (automatically installed when needed): pkgtools/digest: calculates SHA1 checksums (and other kinds) pkgtools/libnbcompat: compat library for pkg tools pkgtools/mtree: installed on non-BSD systems due to lack of native mtree pkgtools/pkg_install: up-to-date replacement for /usr/sbin/pkg_install, or for use on operating systems where pkg_install is not present Utilities used by pkgsrc (not automatically installed): pkgtools/pkg_tarup: create a binary package from an already-installed package. used by 'make replace' to save the old package pkgtools/dfdisk: adds extra functionality to pkgsrc, allowing it to fetch distfiles from multiple locations. It currently supports the following methods: multiple CD-ROMs and network FTP/HTTP connections. pkgtools/xpkgwedge: put X11 packages someplace else (enabled by default) devel/cpuflags: will determine the best compiler flags to optimise code for your current CPU and compiler. Utilities for keeping track of installed packages, being up to date, etc: pkgtools/pkg_chk: installs pkg_chk, which reports on packages whose installed versions do not match the latest pkgsrc entries pkgtools/pkgdep: makes dependency graphs of packages, to aid in choosing a strategy for updating pkgtools/pkgdepgraph: make graph from above (uses graphviz) pkgtools/pkglint: This provides two distinct abilities: check a pkgsrc entry for correctness (pkglint) check for and remove out-of-date distfiles and binary packages (lintpkgsrc) pkgtools/pkgsurvey: report what packages you have installed Utilities for people maintaining or creating individual packages: pkgtools/pkgdiff: automate making and maintaining patches for a package (includes pkgdiff, pkgvi, mkpatches, ...) pkgtools/rpm2pkg, pkgtools/url2pkg: aids in converting to pkgsrc pkgtools/gensolpkg: convert pkgsrc to a Solaris package Utilities for people maintaining pkgsrc (or more obscure pkg utilities) pkgtools/pkgconflict: find packages that conflict but aren't marked as such pkgtools/pkg_comp: build packages in a chrooted area pkgtools/libkver: spoof kernel version for chrooted cross builds How to use pkgsrc as non-root If you want to use pkgsrc as non-root user, you can set some variables to make pkgsrc work under these conditions. Please see this message for more details. How can I install/use XFree86 from pkgsrc? If you want to use XFree86 from pkgsrc instead of your system's own X11 (/usr/X11R6, /usr/openwin, ...), you will have to add the following lines into mk.conf: X11_TYPE=XFree86 How can I install/use X.org from pkgsrc? If you want to use X.org from pkgsrc instead of your system's own X11 (/usr/X11R6, /usr/openwin, ...) you will have to add the following lines into mk.conf: X11_TYPE=xorg How to fetch files from behind a firewall If you are sitting behind a firewall which does not allow direct connections to Internet hosts (i.e. non-NAT), you may specify the relevant proxy hosts. This is done using an environment variable in the form of a URL e.g. in Amdahl, the machine orpheus.amdahl.com is one of the firewalls, and it uses port 80 as the proxy port number. So the proxy environment variables are: ftp_proxy=ftp://orpheus.amdahl.com:80/ http_proxy=http://orpheus.amdahl.com:80/How do I tell make fetch to do passive FTP? This depends on which utility is used to retrieve distfiles. From bsd.pkg.mk, FETCH_CMD is assigned the first available command from the following list: ${LOCALBASE}/bin/ftp /usr/bin/ftp On a default NetBSD installation, this will be /usr/bin/ftp, which automatically tries passive connections first, and falls back to active connections if the server refuses to do passive. For the other tools, add the following to your /etc/mk.conf file: PASSIVE_FETCH=1. Having that option present will prevent /usr/bin/ftp from falling back to active transfers. How to fetch all distfiles at once You would like to download all the distfiles in a single batch from work or university, where you can't run a make fetch. There is an archive of distfiles on ftp.NetBSD.org, but downloading the entire directory may not be appropriate. The answer here is to do a make fetch-list in /usr/pkgsrc or one of it's subdirectories, carry the resulting list to your machine at work/school and use it there If you don't have a NetBSD-compatible ftp(1) (like lukemftp) at work, don't forget to set FETCH_CMD to something that fetches a URL: At home: % cd /usr/pkgsrc % make fetch-list FETCH_CMD=wget DISTDIR=/tmp/distfiles >/tmp/fetch.sh % scp /tmp/fetch.sh work:/tmp At work: % sh /tmp/fetch.sh then tar up /tmp/distfiles and take it home. If you have a machine running NetBSD, and you want to get all distfiles (even ones that aren't for your machine architecture), you can do so by using the above-mentioned make fetch-list approach, or fetch the distfiles directly by running: % make mirror-distfiles If you even decide to ignore NO_{SRC,BIN}_ON_{FTP,CDROM}, then you can get everything by running: % make fetch NO_SKIP=yesWhat does Don't know how to make /usr/share/tmac/tmac.andoc mean? When compiling the pkgtools/pkg_install package, you get the error from make that it doesn't know how to make /usr/share/tmac/tmac.andoc? This indicates that you don't have installed the text set on your machine (nroff, ...). It is recommended to do that to format manpages. In the case of the pkgtools/pkg_install package, you can get away with setting NOMAN=YES either in the environment or in /etc/mk.conf. What does Could not find bsd.own.mk mean? You didn't install the compiler set, comp.tgz, when you installed your NetBSD machine. Please get it and install it, by extracting it in /: # cd / # tar --unlink -zxvpf .../comp.tgz comp.tgz is part of every NetBSD release. Get the one that corresponds to your release (determine via uname -r). Using 'sudo' with pkgsrc When installing packages as non-root user and using the just-in-time su(1) feature of pkgsrc, it can become annoying to type in the root password for each required package installed. To avoid this, the sudo package can be used, which does password caching over a limited time. To use it, install sudo (either as binary package or from security/sudo) and then put the following into your /etc/mk.conf: .if exists(/usr/pkg/bin/sudo) SU_CMD=/usr/pkg/bin/sudo /bin/sh -c .endif Configuration files handling and placement The global variable PKG_SYSCONFBASE (and some others) can be set by the system administrator in /etc/mk.conf to define the place where configuration files get installed. Therefore, packages must be adapted to support this feature. Keep in mind that you should only install files that are strictly necessary in the configuration directory, files that can go to $PREFIX/share should go there. We will take a look at available variables first (bsd.pkg.mk contains more information). PKG_SYSCONFDIR is where the configuration files for a package may be found (that is, the full path, e.g. /etc or /usr/pkg/etc). This value may be customized in various ways: PKG_SYSCONFBASE is the main config directory under which all package configuration files are to be found. Users will typically want to set it to /etc, or accept the default location of $PREFIX/etc. PKG_SYSCONFSUBDIR is the subdirectory of PKG_SYSCONFBASE under which the configuration files for a particular package may be found. Defaults to ${SYSCONFBASE}. PKG_SYSCONFVAR is the special suffix used to distinguish any overriding values for a particular package (see next item). It defaults to ${PKGBASE}, but for a collection of related packages that should all have the same PKG_SYSCONFDIR value, it can be set in each of the package Makefiles to a common value. PKG_SYSCONFDIR.${PKG_SYSCONFVAR} overrides the value of ${PKG_SYSCONFDIR} for packages with the same value for PKG_SYSCONFVAR. As an example, all the various KDE packages may want to set PKG_SYSCONFVAR to kde so admins can set PKG_SYSCONFDIR.kde in /etc/mk.conf to define where to install KDE config files. Programs' configuration directory should be defined during the configure stage. Packages that use GNU autoconf can usually do this by using the --sysconfdir parameter, but this brings some problems as we will see now. When you change this pathname in packages, you should not allow them to install files in that directory directly. Instead they need to install those files under share/examples/${PKGNAME} so PLIST can register them. Once you have the required configuration files in place (under the share/examples directory) the variable CONF_FILES should be set to copy them into PKG_SYSCONFDIR. The contents of this variable is formed by pairs of filenames; the first element of the pair specifies the file inside the examples directory (registered by PLIST) and the second element specifies the target file. This is done this way to allow binary packages to place files in the right directory using INSTALL/DEINSTALL scripts which are created automatically. The package Makefile must also set USE_PKGINSTALL=YES to use these automatically generated scripts. The automatic copying of config files can be toggled by setting the environment variable PKG_CONFIG prior to package installation. Here is an example, taken from mail/mutt/Makefile: EGDIR= ${PREFIX}/share/doc/mutt/samples CONF_FILES= ${EGDIR}/Muttrc ${PKG_SYSCONFDIR}/Muttrc As you can see, this package installs configuration files inside EGDIR, which are registered by PLIST. After that, the variable CONF_FILES lists the installed file first and then the target file. Users will also get an automatic message when files are installed using this method. Automated security checks Please be aware that there can often be bugs in third-party software, and some of these bugs can leave a machine vulnerable to exploitation by attackers. In an effort to lessen the exposure, the NetBSD packages team maintains a database of known-exploits to packages which have at one time been included in pkgsrc. The database can be downloaded automatically, and a security audit of all packages installed on a system can take place. To do this, install the security/audit-packages package. It has two components: download-vulnerability-list, an easy way to download a list of the security vulnerabilities information. This list is kept up to date by the NetBSD security officer and the NetBSD packages team, and is distributed from the NetBSD ftp server: audit-packages, an easy way to audit the current machine, checking each vulnerability which is known. If a vulnerable package is installed, it will be shown by output to stdout, including a description of the type of vulnerability, and a URL containing more information. Use of the audit-packages package is strongly recommended! The following message is displayed as part of the audit-packages installation procedure: === $NetBSD: faq.xml,v 1.1.1.1 2004/10/21 14:27:43 grant Exp $ You may wish to have the vulnerabilities file downloaded daily so that it remains current. This may be done by adding an appropriate entry to the root users crontab(5) entry. For example the entry # download vulnerabilities file 0 3 * * * ${PREFIX}/sbin/download-vulnerability-list >/dev/null 2>&1 will update the vulnerability list every day at 3AM. You may wish to do this more often than once a day. In addition, you may wish to run the package audit from the daily security script. This may be accomplished by adding the following lines to /etc/security.local if [-x ${PREFIX}/sbin/audit-packages]; then ${PREFIX}/sbin/audit-packages fi === The pkgsrc developer's guidePackage components - files, directories and contents Whenever you're preparing a package, there are a number of files involved which are described in the following sections. MakefileBuilding, installation and creation of a binary package are all controlled by the package's Makefile.There is a Makefile for each package. This file includes the standard bsd.pkg.mk file (referenced as ../../mk/bsd.pkg.mk), which sets all the definitions and actions necessary for the package to compile and install itself. The mandatory variables are the DISTNAME which specifies the base name of the distribution file to be downloaded from the site on the Internet, MASTER_SITES which specifies that site, CATEGORIES which denotes the categories into which the package falls, PKGNAME which is the name of the package, the MAINTAINER's name, and the COMMENT variable, which should contain a one-line description of the package (the package name should not appear, it will be added automatically). The maintainer variable is there so that anyone who quibbles with the (always completely correct) decisions taken by the guy who maintains the package can complain vigorously, or send chocolate as a sign of appreciation.The MASTER_SITES may be set to one of the predefined sites: ${MASTER_SITE_APACHE} ${MASTER_SITE_DEBIAN} ${MASTER_SITE_GNOME} ${MASTER_SITE_GNU} ${MASTER_SITE_GNUSTEP} ${MASTER_SITE_IFARCHIVE} ${MASTER_SITE_MOZILLA} ${MASTER_SITE_PERL_CPAN} ${MASTER_SITE_SOURCEFORGE} ${MASTER_SITE_SUNSITE} ${MASTER_SITE_R_CRAN} ${MASTER_SITE_SUSE} ${MASTER_SITE_TEX_CTAN} ${MASTER_SITE_XCONTRIB} ${MASTER_SITE_XEMACS} If one of these predefined sites is chosen, you may require the ability to specify a subdirectory of that site. Since these macros may expand to more than one actual site, you must use the following construct to specify a subdirectory: ${MASTER_SITE_GNU:=subdirectory/name/} ${MASTER_SITE_SOURCEFORGE:=project_name/} Note the trailing slash after the subdirectory name.MASTER_SITE_SUBDIR has been deprecated and should no longer be used.If the package has multiple DISTFILES or multiple PATCHFILES from different sites, set SITES_foo to a list of URI's where file foo may be found. foo includes the suffix, e.g.DISTFILES= ${DISTNAME}${EXTRACT_SUFX} DISTFILES+= foo-file.tar.gz SITES_foo-file.tar.gz=http://www.somewhere.com/somehow/ \ http://www.somewhereelse.com/mirror/somehow/Note that the normal default setting of DISTFILES must be made explicit if you want to add to it (rather than replace it), as you usually would.Currently the following values are available for CATEGORIES. If more than one is used, they need to be separated by spaces: archivers cross geography meta-pkgs security audio databases graphics misc shells benchmarks devel ham multimedia sysutils biology editors inputmethod net textproc cad emulators lang news time chat finance mail parallel wm comms fonts math pkgtools www converters games mbone print x11 Please pay attention to the following gotchas: Add MANCOMPRESSED if manpages are installed in compressed form by the package; see comment in bsd.pkg.mk.Replace /usr/local with ${PREFIX} in all files (see patches, below).If the package installs any info files, see .Set MAINTAINER to be yourself. If you really can't maintain the package for future updates, set it to tech-pkg@NetBSD.org.If a home page for the software in question exists, add the variable HOMEPAGE right after MAINTAINER. The value of this variable should be the URL for the home page.Be sure to set the COMMENT variable to a short description of the package, not containing the pkg's name.distinfoMost important, the mandatory message digest, or checksum, of all the distfiles needed for the package to compile, confirming they match the original file distributed by the author. This ensures that the distfile retrieved from the Internet has not been corrupted during transfer or altered by a malign force to introduce a security hole. It is generated using the make makesum command. The digest algorithm used was, at one stage, md5, but that was felt lacking compared to sha1, and so sha1 is now the default algorithm. The distfile size is also generated and stored in new distinfo files. The pkgtools/digest utility calculates all of the digests in the distinfo file, and it provides various different algorithms. At the current time, the algorithms provided are: md5, rmd160, sha1, sha256, sha384 and sha512.Some packages have different sets of distfiles on a per architecture basis, for example www/navigator). These are kept in the same distinfo file and care should be taken when upgrading such a package to ensure distfile information is not lost.The message digest/checksum for all the official patches found in the patches/ directory (see) for the package is also stored in the distinfo file. This is a message digest/checksum of all lines in the patch file except the NetBSD RCS Id. This file is generated by invoking make makepatchsum (or make mps if you're in a hurry).patches/*This directory contains files that are used by the patch1 command to modify the sources as distributed in the distribution file into a form that will compile and run perfectly on NetBSD. The files are applied successively in alphabetic order (as returned by a shell patches/patch-* glob expansion), so patch-aa is applied before patch-ab, etc.The patch-* files should be in diff -bu format, and apply without a fuzz to avoid problems. (To force patches to apply with fuzz you can set PATCH_FUZZ_FACTOR=-F2). Furthermore, do not put changes for more than one file into a single patch-file, as this will make future modifications more difficult.Similar, a file should be patched at most once, not several times by several different patches. If a file needs several patches, they should be combined into one file.One important thing to mention is to pay attention that no RCS IDs get stored in the patch files, as these will cause problems when later checked into the NetBSD CVS tree. Use the pkgdiff from the pkgtools/pkgdiff package to avoid these problems.For even more automation, we recommend using mkpatches from the same package to make a whole set of patches. You just have to backup files before you edit them to filename.orig, e.g. with cp -p filename filename.orig or, easier, by using pkgvi again from the same package. If you upgrade a package this way, you can easily compare the new set of patches with the previously existing one with patchdiff.When you have finished a package, remember to generate the checksums for the patch files by using the make makepatchsum command, see .Patch files that are distributed by the author or other maintainers can be listed in $PATCHFILES. If it is desired to store any patches that should not be committed into pkgsrc, they can be kept outside the pkgsrc tree in the $LOCALPATCHES directory. The directory tree there is expected to have the same category/package structure as pkgsrc, and patches are expected to be stored inside these dirs (also known as $LOCALPATCHES/$PKGPATH). For example if you want to keep a private patch for pkgsrc/graphics/png, keep it in $LOCALPATCHES/graphics/png/mypatch. All files in the named directory are expected to be patch files, and they are applied after pkgsrc patches are applied.Other mandatory filesDESCRA multi-line description of the piece of software. This should include any credits where they are due. Please bear in mind that others do not share your sense of humour (or spelling idiosyncrasies), and that others will read everything that you write here.PLIST This file governs the files that are installed on your system: all the binaries, manual pages, etc. There are other directives which may be entered in this file, to control the creation and deletion of directories, and the location of inserted files. See for more information. Optional filesINSTALLThis shell script is invoked twice by pkg_add1. First time after package extraction and before files are moved in place, the second time after the files to install are moved in place. This can be used to do any custom procedures not possible with @exec commands in PLIST. See pkg_add1 and pkg_create1 for more information.DEINSTALLThis script is executed before and after any files are removed. It is this script's responsibility to clean up any additional messy details around the package's installation, since all pkg_delete knows is how to delete the files created in the original distribution. See pkg_delete1 and pkg_create1 for more information.MESSAGEDisplay this file after installation of the package. Useful for things like legal notices on almost-free software and hints for updating config files after installing modules for apache, PHP etc. Please note that you can modify variables in it easily by using MESSAGE_SUBST in the package's Makefile:MESSAGE_SUBST+= SOMEVAR="somevalue"replaces "${SOMEVAR}" with somevalue in MESSAGE.work*When you type make the distribution files are unpacked into this directory. It can be removed by running make clean. Besides the sources, this directory is also used to keep various timestamp files. If a package doesn't create a subdirectory for itself (like GNU software does, for instance), but extracts itself in the current directory, you should set WRKSRC accordingly, e.g. editors/sam again, but the quick answer is: WRKSRC= ${WRKDIR} Please note that the old NO_WRKSUBDIR has been deprecated and should not be used. Also, if your package doesn't create a subdir with the name of DISTNAME but some different name, set WRKSRC to point to the proper name in ${WRKDIR}. See lang/tcl and x11/tk for examples, and here is another one: WRKSRC= ${WRKDIR}/${DISTNAME}/unix The name of the working directory created by pkgsrc is work by default. If the same pkgsrc tree should be used on several different platforms, the variable OBJMACHINE can be set in /etc/mk.conf to attach the platform to the directory name, e.g. work.i386 or work.sparc. files/*If you have any files that you wish to be placed in the package prior to configuration or building, you could place these files here and use a ${CP} command in the pre-configure target to achieve this. Alternatively, you could simply diff the file against /dev/null and use the patch mechanism to manage the creation of this file.PLIST issues The PLIST file contains a package's packing list, i.e. a list of files that belong to the package (relative to the ${PREFIX} directory it's been installed in) plus some additional statements - see the pkg_create1 manpage for a full list. This chapter addresses some issues that need attention when dealing with the PLIST file (or files, see below!).RCS ID Be sure to add a RCS ID line as the first thing in any PLIST file you write: @comment $NetBSD$Semi-automatic PLIST generationYou can use the make print-PLIST command to output a PLIST that matches any new files since the package was extracted. See for more information on this target.Tweaking output of make print-PLIST If you have used any of the *-dirs packages, as explained in , you may have noticed that make print-PLIST outputs a set of @comments instead of real @dirrm lines. You can also do this for specific directories and files, so that the results of that command are very close to reality. This helps a lot during the update of packages. The PRINT_PLIST_AWK variable takes a set of AWK patterns and actions that are used to filter the output of print-PLIST. You can append any chunk of AWK scripting you like to it, but be careful with quoting. For example, to get all files inside the libdata/foo directory removed from the resulting PLIST: PRINT_PLIST_AWK+= /^libdata\/foo/ { next; } And to get all the @dirrm lines referring to a specific (shared) directory converted to @comments: PRINT_PLIST_AWK+= /^@dirrm share\/specific/ { print "@comment " $$0; next; } Variable substitution in PLIST A number of variables are substituted automatically in PLISTs when a package is installed on a system. This includes the following variables: ${MACHINE_ARCH}, ${MACHINE_GNU_ARCH}Some packages like emacs and perl embed information about which architecture they were built on into the pathnames where they install their file. To handle this case, PLIST will be preprocessed before actually used, and the symbol ${MACHINE_ARCH} will be replaced by what uname -p gives. The same is done if the string ${MACHINE_GNU_ARCH} is embedded in PLIST somewhere - use this on packages that have GNU autoconf created configure scripts.Legacy noteThere used to be a symbol $ARCH that was replaced by the output of uname -m, but that's no longer supported and has been removed.${OPSYS}, ${LOWER_OPSYS}, ${OS_VERSION}Some packages want to embed the OS name and version into some paths. To do this, use these variables in the PLIST: ${OPSYS} - output of uname -s${LOWER_OPSYS} - lowercase common name (eg. solaris)${OS_VERSION} - uname -r${PKGLOCALEDIR}Packages that install locale files should list them in the PLIST as ${PKGLOCALEDIR}/locale/de/LC_MESSAGES/... instead of share/locale/de/LC_MESSAGES/.... This properly handles the fact that different operating systems expect locale files to be either in share or lib by default. For a complete list of values which are replaced by default, please look in bsd.pkg.mk (and search for PLIST_SUBST). If you want to change other variables not listed above, you can add variables and their expansions to this variable in the following way, similar to MESSAGE_SUBST (see): PLIST_SUBST+= SOMEVAR="somevalue"This replaces all occurrences of ${SOMEVAR} in the PLIST with somevalue. Manpage-compressionManpages should be installed in compressed form if MANZ is set (in bsd.own.mk), and uncompressed otherwise. To handle this in the PLIST file, the suffix .gz is appended/removed automatically for manpages according to MANZ and MANCOMPRESSED being set or not, see above for details. This modification of the PLIST file is done on a copy of it, not PLIST itself.Changing PLIST source with PLIST_SRCTo use one or more files as source for the PLIST used in generating the binary package, set the variable PLIST_SRC to the names of that file(s). The files are later concatenated using cat(1), and order of things is important.Platform specific and differing PLISTsSome packages decide to install a different set of files based on the operating system being used. These differences can be automatically handled by using the following files:PLIST.commonPLIST.${OPSYS}PLIST.common_endIf PLIST.${OPSYS} exists, these files are used instead of PLIST. This allows packages which behave in this way to be handled gracefully. Manually overriding PLIST_SRC for other more exotic uses is also possible.Sharing directories between packages A shared directory is a directory where multiple (and unrelated) packages install files. These directories are problematic because you have to add special tricks in the PLIST to conditionally remove them, or have some centralized package handle them. Within pkgsrc, you'll find both approaches. If a directory is shared by a few unrelated packages, it's often not worth to add an extra package to remove it. Therefore, one simply does: @unexec ${RMDIR} %D/path/to/shared/directory 2>/dev/null || ${TRUE} in the PLISTs of all affected packages, instead of the regular "@dirrm" line. However, if the directory is shared across many packages, two different solutions are available: If the packages have a common dependency, the directory can be removed in that. For example, see textproc/scrollkeeper, which removes the shared directory share/omf. If the packages using the directory are not related at all (they have no common dependencies), a *-dirs package is used. From now on, we'll discuss the second solution. To get an idea of the *-dirs packages available, issue: % cd .../pkgsrc % ls -d */*-dirs Their use from other packages is very simple. The USE_DIRS variable takes a list of package names (without the -dirs part) together with the required version number (always pick the latest one when writting new packages). For example, if a package installs files under share/applications, it should have the following line in it: USE_DIRS+= xdg-1.1 After regenerating the PLIST using make print-PLIST, you should get the right (commented out) lines. Note that, even if your package is using $X11BASE, it must not depend on the *-x11-dirs packages. Just specify the name without that part and pkgsrc (in particular, mk/dirs.mk) will take care of it. Buildlink methodologyBuildlink is a framework in pkgsrc that controls what headers and libraries are seen by a package's configure and build processes. This is implemented in a two step process:Symlink headers and libraries for dependencies into BUILDLINK_DIR, which by default is a subdirectory of WRKDIR.Create wrapper scripts that are used in place of the normal compiler tools that translate -I${LOCALBASE}/include and -L${LOCALBASE}/lib into references to BUILDLINK_DIR. The wrapper scripts also make native compiler on some operating systems look like GCC, so that packages that expect GCC won't require modifications to build with those native compilers.This normalizes the environment in which a package is built so that the package may be built consistently despite what other software may be installed. Please note that the normal system header and library paths, e.g. /usr/include, /usr/lib, etc., are always searched -- buildlink3 is designed to insulate the package build from non-system-supplied software. Converting packages to use buildlink3The process of converting packages to use the buildlink3 framework (bl3ifying) is fairly straightforward. The things to keep in mind are:Set USE_BUILDLINK3 to yes. Ensure that the build always calls the wrapper scripts instead of the actual toolchain. Some packages are tricky, and the only way to know for sure is the check ${WRKDIR}/.work.log to see if the wrappers are being invoked. Don't override PREFIX from within the package Makefile, e.g. Java VMs, standalone shells, etc., because the code to symlink files into ${BUILDLINK_DIR} looks for files relative to pkg_info -qp pkgname. Remember that only the buildlink3.mk files that you list in a package's Makefile are added as dependencies for that package. If a dependency on a particular package is required for its libraries and headers, then we replace: DEPENDS+= foo>=1.1.0:../../category/foowith.include "../../category/foo/buildlink3.mk"There are several buildlink3.mk files in pkgsrc/mk that handle special package issues:bdb.buildlink3.mk chooses either the native or a pkgsrc Berkeley DB implementation based on the values of BDB_ACCEPTED and BDB_DEFAULT.curses.buildlink3.mk If the system comes with neither Curses nor NCurses, this will take care to install the devel/ncurses package.krb5.buildlink3.mk uses the value of KRB5_ACCEPTED to choose between adding a dependency on Heimdal or MIT-krb5 for packages that require a Kerberos 5 implementation.motif.buildlink3.mk checks for a system-provided Motif installation or adds a dependency on x11/lesstif or x11/openmotif;ossaudio.buildlink3.mk defines several variables that may be used by packages that use the Open Sound System (OSS) API;pgsql.buildlink3.mk will accept either Postgres 7.3 or 7.4, whichever is found installed. See the file for more information. pthread.buildlink3.mk uses the value of PTHREAD_OPTS and checks for native pthreads or adds a dependency on devel/pth as needed;xaw.buildlink3.mk uses the value of XAW_TYPE to choose a particular Athena widgets library.The comments in those buildlink3.mk files provide a more complete description of how to use them properly.Writing buildlink3.mk files A package's buildlink3.mk file is included by Makefiles to indicate the need to compile and link against header files and libraries provided by the package. A buildlink3.mk file should always provide enough information to add the correct type of dependency relationship and include any other buildlink3.mk files that it needs to find headers and libraries that it needs in turn. To generate an initial buildlink3.mk file for further editing, Rene Hexel's pkgtools/createbuildlink package is highly recommended. For most packages, the following command will generate a good starting point for buildlink3.mk files:% cd pkgsrc/category/pkgdir % createbuildlink -3 >buildlink3.mkAnatomy of a buildlink3.mk fileThe following real-life example buildlink3.mk is taken from pkgsrc/graphics/tiff: # $NetBSD: buildlink3.mk,v 1.7 2004/03/18 09:12:12 jlam Exp $ BUILDLINK_DEPTH:= ${BUILDLINK_DEPTH}+ TIFF_BUILDLINK3_MK:= ${TIFF_BUILDLINK3_MK}+ .if !empty(BUILDLINK_DEPTH:M+) BUILDLINK_DEPENDS+= tiff .endif BUILDLINK_PACKAGES:= ${BUILDLINK_PACKAGES:Ntiff} BUILDLINK_PACKAGES+= tiff .if !empty(TIFF_BUILDLINK3_MK:M+) BUILDLINK_DEPENDS.tiff+= tiff>=3.6.1 BUILDLINK_PKGSRCDIR.tiff?= ../../graphics/tiff .endif # TIFF_BUILDLINK3_MK .include "../../devel/zlib/buildlink3.mk" .include "../../graphics/jpeg/buildlink3.mk" BUILDLINK_DEPTH:= ${BUILDLINK_DEPTH:S/+$//} The header and footer manipulate BUILDLINK_DEPTH, which is common across all buildlink3.mk files and is used to track at what depth we are including buildlink3.mk files. The first section controls if the dependency on pkg is added. BUILDLINK_DEPENDS is the global list of packages for which dependencies are added by buildlink3. The second section advises pkgsrc that the buildlink3.mk file for pkg has been included at some point. BUILDLINK_PACKAGES is the global list of packages for which buildlink3.mk files have been included. It must always be appended to within a buildlink3.mk file. The third section is protected from multiple inclusion and controls how the dependency on pkg is added. Several important variables are set in the section: BUILDLINK_DEPENDS.pkg is the actual dependency recorded in the installed package; this should always be set using += to ensure that we're appending to any pre-existing list of values. This variable should be set to the first version of the package that had the last change in the major number of a shared library or that had a major API change. BUILDLINK_PKGSRCDIR.pkg is the location of the pkg pkgsrc directory; BUILDLINK_DEPMETHOD.pkg (not shown above) controls whether we use BUILD_DEPENDS or DEPENDS to add the dependency on pkg. The build dependency is selected by setting BUILDLINK_DEPMETHOD.pkg to build. By default, the full dependency is used. BUILDLINK_INCDIRS.pkg and BUILDLINK_LIBDIRS.pkg (not shown above) are lists of subdirectories of ${BUILDLINK_PREFIX.pkg} to add to the header and library search paths. These default to include and lib respectively. BUILDLINK_CPPFLAGS.pkg (not shown above) is the list of preprocessor flags to add to CPPFLAGS, which are passed on to the configure and build phases. The -I option should be avoided and instead be handled using BUILDLINK_INCDIRS.pkg as above. The following variables are all optionally defined within this second section (protected against multiple inclusion) and control which package files are symlinked into ${BUILDLINK_DIR} and how their names are transformed during the symlinking: BUILDLINK_FILES.pkg (not shown above) is a shell glob pattern relative to ${BUILDLINK_PREFIX.pkg} to be symlinked into ${BUILDLINK_DIR}, e.g. include/*.h. BUILDLINK_FILES_CMD.pkg (not shown above) is a shell pipeline that outputs to stdout a list of files relative to ${BUILDLINK_PREFIX.pkg}. The resulting files are to be symlinked into ${BUILDLINK_DIR}. By default, this takes the +CONTENTS of a pkg and filters it through ${BUILDLINK_CONTENTS_FILTER.pkg}. BUILDLINK_CONTENTS_FILTER.pkg (not shown above) is a filter command that filters +CONTENTS input into a list of files relative to ${BUILDLINK_PREFIX.pkg} on stdout. By default for overwrite packages, BUILDLINK_CONTENTS_FILTER.pkg outputs the contents of the include and lib directories in the package +CONTENTS, and for pkgviews packages, it outputs any libtool archives in lib directories. BUILDLINK_TRANSFORM.pkg (not shown above) is a list of sed arguments used to transform the name of the source filename into a destination filename, e.g. -e "s|/curses.h|/ncurses.h|g". The last section includes any buildlink3.mk needed for pkg's library dependencies. Including these buildlink3.mk files means that the headers and libraries for these dependencies are also symlinked into ${BUILDLINK_DIR} whenever the pkg buildlink3.mk file is included. Updating BUILDLINK_DEPENDS.pkg in buildlink3.mk files There are two situations that require increasing the dependency listed in BUILDLINK_DEPENDS.pkg after a package update: if the sonames (major number of the library version) of any installed shared libraries change; if the API or interface to the header files change. In these cases, BUILDLINK_DEPENDS.pkg should be adjusted to require at least the new package version. In some cases, the packages that depend on this new version may need their PKGREVISIONs increased and, if they have buildlink3.mk files, their BUILDLINK_DEPENDS.pkg adjusted, too. This is needed so that binary packages made using it will require the correct package dependency and not settle for an older one which will not contain the necessary shared libraries. Please take careful consideration before adjusting BUILDLINK_DEPENDS.pkg as we don't want to cause unneeded package deletions and rebuilds. In many cases, new versions of packages work just fine with older dependencies. See and for more information about dependencies on other packages, including the BUILDLINK_RECOMMENDED and RECOMMENDED definitions. Writing builtin.mk files Some packages in pkgsrc install headers and libraries that coincide with headers and libraries present in the base system. Aside from a buildlink3.mk file, these packages should also include a builtin.mk file that includes the necessary checks to decide whether using the built-in software or the pkgsrc software is appropriate. The only requirements of a builtin.mk file for pkg are: It should set USE_BUILTIN.pkg to either yes or no after it is included. It should not override any USE_BUILTIN.pkg which is already set before the builtin.mk file is included. It should be written to allow multiple inclusion. This is very important and takes careful attention to Makefile coding. Anatomy of a builtin.mk fileThe following is the recommended template for builtin.mk files: .if !defined(IS_BUILTIN.foo) # # IS_BUILTIN.foo is set to "yes" or "no" depending on whether "foo" # genuinely exists in the system or not. # IS_BUILTIN.foo?= no # BUILTIN_PKG.foo should be set here if "foo" is built-in and its package # version can be determined. # . if !empty(IS_BUILTIN.foo:M[yY][eE][sS]) BUILTIN_PKG.foo?= foo-1.0 . endif .endif # IS_BUILTIN.foo .if !defined(USE_BUILTIN.foo) USE_BUILTIN.foo?= ${IS_BUILTIN.foo} . if defined(BUILTIN_PKG.foo) . for _depend_ in ${BUILDLINK_DEPENDS.foo} . if !empty(USE_BUILTIN.foo:M[yY][eE][sS]) USE_BUILTIN.foo!= \ if ${PKG_ADMIN} pmatch '${_depend_}' ${BUILTIN_PKG.foo}; then \ ${ECHO} "yes"; \ else \ ${ECHO} "no"; \ fi . endif . endfor . endif .endif # USE_BUILTIN.foo CHECK_BUILTIN.foo?= no .if !empty(CHECK_BUILTIN.foo:M[nN][oO]) # # Here we place code that depends on whether USE_BUILTIN.foo is set to # "yes" or "no". # .endif # CHECK_BUILTIN.foo The first section sets IS_BUILTIN.pkg depending on if pkg really exists in the base system. This should not be a base system software with similar functionality to pkg; it should only be yes if the actual package is included as part of the base system. This variable is only used internally within the builtin.mk file. The second section sets BUILTIN_PKG.pkg to the version of pkg in the base system if it exists (if IS_BUILTIN.pkg is yes). This variable is only used internally within the builtin.mk file. The third section sets USE_BUILTIN.pkg and is required in all builtin.mk files. The code in this section must make the determination whether the built-in software is adequate to satisfy the dependencies listed in BUILDLINK_DEPENDS.pkg. This is typically done by comparing BUILTIN_PKG.pkg against each of the dependencies in BUILDLINK_DEPENDS.pkg. USE_BUILTIN.pkg must be set to the correct value by the end of the builtin.mk file. Note that USE_BUILTIN.pkg may be yes even if IS_BUILTIN.pkg is no because we may make the determination that the built-in version of the software is similar enough to be used as a replacement. The last section is guarded by CHECK_BUILTIN.pkg, and includes code that uses the value of USE_BUILTIN.pkg set in the previous section. This typically includes, e.g., adding additional dependency restrictions and listing additional files to symlink into ${BUILDLINK_DIR} (via BUILDLINK_FILES.pkg). Global preferences for native or pkgsrc software When building packages, it's possible to choose whether to set a global preference for using either the built-in (native) version or the pkgsrc version of software to satisfy a dependency. This is controlled by setting PREFER_PKGSRC and PREFER_NATIVE. These variables take values of either yes, no, or a list of packages. PREFER_PKGSRC tells pkgsrc to use the pkgsrc versions of software, while PREFER_NATIVE tells pkgsrc to use the built-in versions. Preferences are determined by the most specific instance of the package in either PREFER_PKGSRC or PREFER_NATIVE. If a package is specified in neither or in both variables, then PREFER_PKGSRC has precedence over PREFER_NATIVE. For example, to require using pkgsrc versions of software for all but the most basic bits on a NetBSD system, you can set: PREFER_PKGSRC= yes PREFER_NATIVE= getopt skey tcp_wrappers A package must have a builtin.mk file to be listed in PREFER_NATIVE, otherwise it is simply ignored in that list. Options handling Many packages have the ability to be built to support different sets of features. bsd.options.mk is a framework in pkgsrc that provides generic handling of those options that determine different ways in which the packages can be built. It's possible for the user to specify exactly which sets of options will be built into a package or to allow a set of global default options apply. Global default options Global default options are listed in PKG_DEFAULT_OPTIONS, which is a list of the options that should be built into every package if that option is supported. This variable should be set in /etc/mk.conf. Converting packages to use bsd.options.mk The following example shows how bsd.options.mk should be use in a package Makefile, or in a file, e.g. options.mk, that is included by the main package Makefile. # Global and legacy options .if defined(WIBBLE_USE_OPENLDAP) && !empty(WIBBLE_USE_OPENLDAP:M[yY][eE][sS]) PKG_DEFAULT_OPTIONS+= ldap .endif .if defined(USE_SASL2) && !empty(USE_SASL2:M[yY][eE][sS]) PKG_DEFAULT_OPTIONS+= sasl .endif PKG_OPTIONS_VAR= PKG_OPTIONS.wibble PKG_SUPPORTED_OPTIONS= ldap sasl # # Default options for "wibble" package. # .if !defined(PKG_OPTIONS.wibble) PKG_DEFAULT_OPTIONS+= sasl endif .include "../../mk/bsd.options.mk" # Package-specific option-handling ### ### LDAP support ### .if !empty(PKG_OPTIONS:Mldap) . include "../../databases/openldap/buildlink3.mk" CONFIGURE_ARGS+= --enable-ldap=${BUILDLINK_PREFIX.openldap} .endif ### ### SASL authentication ### .if !empty(PKG_OPTIONS:Msasl) . include "../../security/cyrus-sasl2/buildlink3.mk" CONFIGURE_ARGS+= --enable-sasl=${BUILDLINK_PREFIX.sasl} .endif The first section only exists if you are converting a package that had its own ad-hoc options handling to use bsd.options.mk. It converts global or legacy options variables into an equivalent PKG_OPTIONS.pkg value. These sections will be removed over time as the old options are in turn deprecated and removed. The second section contains the information about which build options are supported by the package, and any default options settings if needed. PKG_OPTIONS_VAR is a list of the name of the make1 variables that contain the options the user wishes to select. The recommended value is PKG_OPTIONS.pkg but any package-specific value may be used. This variable should be set in a package Makefile. PKG_SUPPORTED_OPTIONS is a list of build options supported by the package. This variable should be set in a package Makefile. ${PKG_OPTIONS_VAR} (the variables named in PKG_OPTIONS_VAR) are variables that list the selected build options and override any default options given in PKG_DEFAULT_OPTIONS. If any of the options begin with a -, then that option is always removed from the selected build options, e.g. PKG_DEFAULT_OPTIONS= kerberos ldap sasl PKG_OPTIONS_VAR= WIBBLE_OPTIONS WIBBLE_OPTIONS= ${PKG_DEFAULT_OPTIONS} -sasl # implies PKG_OPTIONS == "kerberos ldap" or PKG_OPTIONS_VAR= WIBBLE_OPTIONS WIBBLE_OPTIONS= kerberos -ldap ldap # implies PKG_OPTIONS == "kerberos" This variable should be set in /etc/mk.conf. After the inclusion of bsd.options.mk, the following variables are set: PKG_OPTIONS contains the list of the selected build options, properly filtered to remove unsupported and duplicate options. The remaining sections contain the logic that is specific to each option. There should be a check for every option listed in PKG_SUPPORTED_OPTIONS, and there should be clear documentation on what turning on the option will do in the comments preceding each section. The correct way to check for an option is to check whether it is listed in PKG_OPTIONS. The build processThe basic steps for building a program are always the same. First the program's source (distfile) must be brought to the local system and then extracted. After any patches to compile properly on NetBSD are applied, the software can be configured, then built (usually by compiling), and finally the generated binaries, etc. can be put into place on the system. These are exactly the steps performed by the NetBSD package system, which is implemented as a series of targets in a central Makefile, pkgsrc/mk/bsd.pkg.mk.Program locationBefore outlining the process performed by the NetBSD package system in the next section, here's a brief discussion on where programs are installed, and which variables influence this.The automatic variable PREFIX indicates where all files of the final program shall be installed. It is usually set to LOCALBASE (/usr/pkg), or CROSSBASE for pkgs in the cross category. The value of PREFIX needs to be put into the various places in the program's source where paths to these files are encoded. See and for more details.When choosing which of these variables to use, follow the following rules:PREFIX always points to the location where the current pkg will be installed. When referring to a pkg's own installation path, use ${PREFIX}.LOCALBASE is where all non-X11 pkgs are installed. If you need to construct a -I or -L argument to the compiler to find includes and libraries installed by another non-X11 pkg, use ${LOCALBASE}.X11BASE is where the actual X11 distribution (from xsrc, etc.) is installed. When looking for standard X11 includes (not those installed by a pkg), use ${X11BASE}.X11 based are special in that they may be installed in either X11BASE or LOCALBASE.Usually, X11 packages should be installed under LOCALBASE whenever possible. Note that you will need to set USE_X11 in them to request the presence of X11 and to get the right compilation flags.Even though, there are some packages that cannot be installed under LOCALBASE: those that come with app-defaults files. These packages are special and they must be placed under X11BASE. To accomplish this, set either USE_X11BASE or USE_IMAKE in your package.Some notes: USE_X11 and USE_X11BASE are mutually exclusive. If you need to find includes or libraries installed by a pkg that has USE_IMAKE or USE_X11BASE in its pkg Makefile, you need to use both ${X11BASE} and

${LOCALBASE}. To force installation of all X11 packages in LOCALBASE, the pkgtools/xpkgwedge is enabled by default.X11PREFIX should be used to refer to the installed location of an X11 package. X11PREFIX will be set to X11BASE if xpkgwedge is not installed, and to LOCALBASE if xpkgwedge is installed.If xpkgwedge is installed, it is possible to have some packages installed in X11BASE and some in LOCALBASE. To determine the prefix of an installed package, the EVAL_PREFIX definition can be used. It takes pairs in the format DIRNAME=<package>, and the make(1) variable DIRNAME will be set to the prefix of the installed package <package>, or ${X11PREFIX} if the package is not installed.This is best illustrated by example.The following lines are taken from pkgsrc/wm/scwm/Makefile:EVAL_PREFIX+= GTKDIR=gtk+ CONFIGURE_ARGS+= --with-guile-prefix=${LOCALBASE} \ --with-gtk-prefix="${GTKDIR}" \ --enable-multibyteSpecific defaults can be defined for the packages evaluated using EVAL_PREFIX, by using a definition of the form:GTKDIR_DEFAULT= ${LOCALBASE}where GTKDIR corresponds to the first definition in the EVAL_PREFIX pair.Within ${PREFIX}, packages should install files according to hier(7), with the exception that manual pages go into ${PREFIX}/man, not ${PREFIX}/share/man.Main targetsThe main targets used during the build process defined in bsd.pkg.mk are:fetchThis will check if the file(s) given in the variables DISTFILES and PATCHFILES (as defined in the package's Makefile) are present on the local system in /usr/pkgsrc/distfiles. If they are not present, an attempt will be made to fetch them using commands of the form:${FETCH_CMD} ${FETCH_BEFORE_ARGS} ${site}${file} ${FETCH_AFTER_ARGS}where ${site} varies through several possibilities in turn: first, MASTER_SITE_OVERRIDE is tried, then the sites specified in either SITES_file if defined, else MASTER_SITES or PATCH_SITES, as applies, then finally the value of MASTER_SITE_BACKUP. The order of all except the first can be optionally sorted by the user, via setting either MASTER_SORT_AWK or MASTER_SORT_REGEX.checksumAfter the distfile(s) are fetched, their checksum is generated and compared with the checksums stored in the distinfo file. If the checksums don't match, the build is aborted. This is to ensure the same distfile is used for building, and that the distfile wasn't changed, e.g. by some malign force, deliberately changed distfiles on the master distribution site or network lossage.extractWhen the distfiles are present on the local system, they need to be extracted, as they are usually in the form of some compressed archive format, most commonly .tar.gz. If only some of the distfiles need to be uncompressed, the files to be uncompressed should be put into EXTRACT_ONLY. If the distfiles are not in .tar.gz format, they can be extracted by setting either EXTRACT_SUFX, or EXTRACT_CMD, EXTRACT_BEFORE_ARGS and EXTRACT_AFTER_ARGS. In the former case, pkgsrc knows how to extract a number of suffixes (.tar.gz, .tgz, .tar.gz2, .tbz, .tar.Z, .tar, .shar.gz, .shar.bz2, .shar.Z, .shar, .Z, .bz2 and .gz; see the definition of the various DECOMPRESS_CMD variables bsd.pkg.mk for a complete list). Here's an example on how to use the other variables for a program that comes with a compressed shell archive whose name ends in .msg.gz: EXTRACT_SUFX= .msg.gz EXTRACT_CMD= zcat EXTRACT_BEFORE_ARGS= EXTRACT_AFTER_ARGS= |shpatchAfter extraction, all the patches named by the PATCHFILES, those present in the patches subdirectory of the package as well as in $LOCALPATCHES/$PKGPATH (e.g. /usr/local/patches/graphics/png) are applied. Patchfiles ending in .Z or .gz are uncompressed before they are applied, files ending in .orig or .rej are ignored. Any special options to patch(1) can be handed in PATCH_DIST_ARGS. See for more details.By default patch(1) is given special args to make it fail if the patches apply with some lines of fuzz. Please fix (regen) the patches so that they apply cleanly. The rationale behind this is that patches that don't apply cleanly may end up being applied in the wrong place, and cause severe harm there.configureMost pieces of software need information on the header files, system calls, and library routines which are available in NetBSD. This is the process known as configuration, and is usually automated. In most cases, a script is supplied with the source, and its invocation results in generation of header files, Makefiles, etc.If the program's distfile contains its own configure script, this can be invoked by setting HAS_CONFIGURE. If the configure script is a GNU autoconf script, GNU_CONFIGURE should be specified instead. In either case, any arguments to the configure script can be specified in the CONFIGURE_ARGS variable, and the configure script's name can be set in CONFIGURE_SCRIPT if it differs from the default configure. Here's an example from the sysutils/top package:HAS_CONFIGURE= yes CONFIGURE_SCRIPT= Configure CONFIGURE_ARGS+= netbsd13If the program uses an Imakefile for configuration, the appropriate steps can be invoked by setting USE_IMAKE to YES. (If you only want the package installed in $X11PREFIX but xmkmf not being run, set USE_X11BASE instead!)buildOnce configuration has taken place, the software will be built by invoking $MAKE_PROGRAM on $MAKEFILE with $BUILD_TARGET as the target to build. The default MAKE_PROGRAM is gmake if USE_GNU_TOOLS contains make, make otherwise. MAKEFILE is set to Makefile by default, and BUILD_TARGET defaults to all. Any of these variables can be set in the package's Makefile to change the default build process.installOnce the build stage has completed, the final step is to install the software in public directories, so users can access the programs and files. As in the build-target, $MAKE_PROGRAM is invoked on $MAKEFILE here, but with the $INSTALL_TARGET instead, the latter defaulting to install (plus install.man, if USE_IMAKE is set).If no target is specified, the default is build. If a subsequent stage is requested, all prior stages are made: e.g. make build will also perform the equivalent of: make fetch make checksum make extract make patch make configure make buildOther helpful targetspre/post-*For any of the main targets described in the previous section, two auxiliary targets exist with pre- and post- used as a prefix for the main target's name. These targets are invoked before and after the main target is called, allowing extra configuration or installation steps be performed from a package's Makefile, for example, which a program's configure script or install target omitted.do-*Should one of the main targets do the wrong thing, and should there be no variable to fix this, you can redefine it with the do-* target. (Note that redefining the target itself instead of the do-* target is a bad idea, as the pre-* and post-* targets won't be called anymore, etc.) You will not usually need to do this.reinstallIf you did a make install and you noticed some file was not installed properly, you can repeat the installation with this target, which will ignore the already installed flag.deinstallThis target does a pkg_delete(1) in the current directory, effectively de-installing the package. The following variables can be used to tune the behaviour:PKG_VERBOSEAdd a "-v" to the pkg_delete1 command.DEINSTALLDEPENDSRemove all packages that require (depend on) the given package. This can be used to remove any packages that may have been pulled in by a given package, e.g. if make deinstall DEINSTALLDEPENDS=1 is done in pkgsrc/x11/kde, this is likely to remove whole KDE. Works by adding -R to the pkg_delete1 command line.updateThis target causes the current package to be updated to the latest version. The package and all depending packages first get de-installed, then current versions of the corresponding packages get compiled and installed. This is similar to manually noting which packages are currently installed, then performing a series of make deinstall and make install (or whatever UPDATE_TARGET is set to) for these packages.You can use the update target to resume package updating in case a previous make update was interrupted for some reason. However, in this case, make sure you don't call make clean or otherwise remove the list of dependent packages in WRKDIR. Otherwise you lose the ability to automatically update the current package along with the dependent packages you have installed.Resuming an interrupted make update will only work as long as the package tree remains unchanged. If the source code for one of the packages to be updated has been changed, resuming make update will most certainly fail!The following variables can be used either on the command line or in /etc/mk.conf to alter the behaviour of make update:UPDATE_TARGETInstall target to recursively use for the updated package and the dependent packages. Defaults to DEPENDS_TARGET if set, install otherwise for make update. e.g. make update UPDATE_TARGET=packageNOCLEANDon't clean up after updating. Useful if you want to leave the work sources of the updated packages around for inspection or other purposes. Be sure you eventually clean up the source tree (see the clean-update target below) or you may run into troubles with old source code still lying around on your next make or make update.REINSTALLDeinstall each package before installing (making DEPENDS_TARGET). This may be necessary if the clean-update target (see below) was called after interrupting a running make update.DEPENDS_TARGETAllows you to disable recursion and hardcode the target for packages. The default is update for the update target, facilitating a recursive update of prerequisite packages. Only set DEPENDS_TARGET if you want to disable recursive updates. Use UPDATE_TARGET instead to just set a specific target for each package to be installed during make update (see above).clean-updateClean the source tree for all packages that would get updated if make update was called from the current directory. This target should not be used if the current package (or any of its depending packages) have already been de-installed (e.g., after calling make update) or you may lose some packages you intended to update. As a rule of thumb: only use this target before the first time you run make update and only if you have a dirty package tree (e.g., if you used NOCLEAN).If you unsure about whether your tree is clean you can either perform a make clean at the top of the tree, or use the following sequence of commands from the directory of the package you want to update (before running make update for the first time, otherwise you lose all the packages you wanted to update!):# make clean-update # make clean CLEANDEPENDS=YES # make updateThe following variables can be used either on the command line or in /etc/mk.conf to alter the behaviour of make clean-update:CLEAR_DIRLISTAfter make clean, do not reconstruct the list of directories to update for this package. Only use this if make update successfully installed all packages you wanted to update. Normally, this is done automatically on make update, but may have been suppressed by the NOCLEAN variable (see above).infoThis target invokes pkg_info1 for the current package. You can use this to check which version of a package is installed.readmeThis target generates a README.html file, which can be viewed using a browser such as www/mozilla or www/links. The generated files contain references to any packages which are in the PACKAGES directory on the local host. The generated files can be made to refer to URLs based on FTP_PKG_URL_HOST and FTP_PKG_URL_DIR. For example, if I wanted to generate README.html files which pointed to binary packages on the local machine, in the directory /usr/packages, set FTP_PKG_URL_HOST=file://localhost and FTP_PKG_URL_DIR=/usr/packages. The ${PACKAGES} directory and its subdirectories will be searched for all the binary packages.readme-allUse this target to create a file README-all.html which contains a list of all packages currently available in the NetBSD Packages Collection, together with the category they belong to and a short description. This file is compiled from the pkgsrc/*/README.html files, so be sure to run this after a make readme.cdrom-readmeThis is very much the same as the readme target (see above), but is to be used when generating a pkgsrc tree to be written to a CD-ROM. This target also produces README.html files, and can be made to refer to URLs based on CDROM_PKG_URL_HOST and CDROM_PKG_URL_DIR.show-distfilesThis target shows which distfiles and patchfiles are needed to build the package. (DISTFILES and PATCHFILES, but not patches/*)show-downlevelThis target shows nothing if the package is not installed. If a version of this package is installed, but is not the version provided in this version of pkgsrc, then a warning message is displayed. This target can be used to show which of your installed packages are downlevel, and so the old versions can be deleted, and the current ones added.show-pkgsrc-dirThis target shows the directory in the pkgsrc hierarchy from which the package can be built and installed. This may not be the same directory as the one from which the package was installed. This target is intended to be used by people who may wish to upgrade many packages on a single host, and can be invoked from the top-level pkgsrc Makefile by using the show-host-specific-pkgs target.show-installed-dependsThis target shows which installed packages match the current package's DEPENDS. Useful if out of date dependencies are causing build problems.check-shlibsAfter a package is installed, check all its binaries and (on ELF platforms) shared libraries to see if they find the shared libs they need. Run by default if PKG_DEVELOPER is set in /etc/mk.conf.print-PLISTAfter a make install from a new or upgraded pkg, this prints out an attempt to generate a new PLIST from a find -newer work/.extract_done. An attempt is made to care for shared libs etc., but it is strongly recommended to review the result before putting it into PLIST. On upgrades, it's useful to diff the output of this command against an already existing PLIST file.If the package installs files via tar(1) or other methods that don't update file access times, be sure to add these files manually to your PLIST, as the find -newer command used by this target won't catch them! See for more information on this target.bulk-packageUsed to do bulk builds. If an appropriate binary package already exists, no action is taken. If not, this target will compile, install and package it (and it's depends, if PKG_DEPENDS is set properly. See . After creating the binary package, the sources, the just-installed package and it's required packages are removed, preserving free disk space.Beware that this target may deinstall all packages installed on a system!bulk-installUsed during bulk-installs to install required packages. If an upto-date binary package is available, it will be installed via pkg_add1. If not, make bulk-package will be executed, but the installed binary not be removed. A binary package is considered upto-date to be installed via pkg_add1 if:None of the package's files (Makefile, ...) were modified since it was built.None of the package's required (binary) packages were modified since it was built.Beware that this target may deinstall all packages installed on a system!Notes on fixes for packagesGeneral operationHow to pull in variables from /etc/mk.conf The problem with package-defined variables that can be overridden via MAKECONF or /etc/mk.conf is that make1 expands a variable as it is used, but evaluates preprocessor like statements (.if, .ifdef and .ifndef) as they are read. So, to use any variable (which may be set in /etc/mk.conf) in one of the .if* statements, the file /etc/mk.conf must be included before that .if* statement. Rather than have a number of ad-hoc ways of including /etc/mk.conf, should it exist, or MAKECONF, should it exist, include the pkgsrc/mk/bsd.prefs.mk file in the package Makefile before any preprocessor-like .if, .ifdef, or .ifndef statements: .include "../../mk/bsd.prefs.mk" .if defined(USE_MENUS) ... .endif If you wish to set the CFLAGS variable in /etc/mk.conf please make sure to use: CFLAGS+= -your -flags Using CFLAGS= (i.e. without the +) may lead to problems with packages that need to add their own flags. Also, you may want to take a look at the devel/cpuflags package if you're interested in optimization for the current CPU. Restricted packages Some licenses restrict how software may be re-distributed. In order to satisfy these restrictions, the package system defines five make variables that can be set to note these restrictions: RESTRICTED This variable should be set whenever a restriction exists (regardless of its kind). Set this variable to a string containing the reason for the restriction. NO_BIN_ON_CDROM Binaries may not be placed on CD-ROM. Set this variable to ${RESTRICTED} whenever a binary package may not be included on a CD-ROM. NO_BIN_ON_FTP Binaries may not be placed on an FTP server. Set this variable to ${RESTRICTED} whenever a binary package may not not be made available on the Internet. NO_SRC_ON_CDROM Distfiles may not be placed on CD-ROM. Set this variable to ${RESTRICTED} if re-distribution of the source code or other distfile(s) is not allowed on CD-ROMs. NO_SRC_ON_FTP Distfiles may not be placed on FTP. Set this variable to ${RESTRICTED} if re-distribution of the source code or other distfile(s) via the Internet is not allowed. Please note that the use of NO_PACKAGE, IGNORE, NO_CDROM, or other generic make variables to denote restrictions is deprecated, because they unconditionally prevent users from generating binary packages! Handling dependencies Your package may depend on some other package being present - and there are various ways of expressing this dependency. pkgsrc supports the BUILD_DEPENDS and DEPENDS definitions, as well as dependencies via buildlink3.mk, which is the preferred way to handle dependencies, and which uses the variables named above. See for more information. The basic difference between the two variables is as follows: The DEPENDS definition registers that pre-requisite in the binary package so it will be pulled in when the binary package is later installed, whilst the BUILD_DEPENDS definition does not, marking a dependency that is only needed for building the package. This means that if you only need a package present whilst you are building, it should be noted as a BUILD_DEPENDS. The format for a BUILD_DEPENDS and a DEPENDS definition is: <pre-req-package-name>:../../<category>/<pre-req-package> Please note that the pre-req-package-name may include any of the wildcard version numbers recognised by pkg_info1. If your package needs another package's binaries or libraries to build or run, and if that package has a buildlink3.mk file available, use it: .include "../../graphics/jpeg/buildlink3.mk" If your package needs to use another package to build itself and there is no buildlink3.mk file available, use the BUILD_DEPENDS definition: BUILD_DEPENDS+= autoconf-2.13:../../devel/autoconf If your package needs a library with which to link and again there is no buildlink3.mk file available, this is specified using the DEPENDS definition. An example of this is the print/lyx package, which uses the xpm library, version 3.4j to build: DEPENDS+= xpm-3.4j:../../graphics/xpm You can also use wildcards in package dependences: DEPENDS+= xpm-[0-9]*:../../graphics/xpm Note that such wildcard dependencies are retained when creating binary packages. The dependency is checked when installing the binary package and any package which matches the pattern will be used. Wildcard dependencies should be used with care. The -[0-9]* should be used instead of -* to avoid potentially ambiguous matches such as tk-postgresql matching a tk-* DEPENDS. Wildcards can also be used to specify that a package will only build against a certain minimum version of a pre-requisite: DEPENDS+= tiff>=3.5.4:../../graphics/tiff This means that the package will build against version 3.5.4 of the tiff library or newer. Such a dependency may be warranted if, for example, the API of the library has changed with version 3.5.4 and a package would not compile against an earlier version of tiff. Please note that such dependencies should only be updated if a package requires a newer pre-requisite, but not to denote recommendations such as security updates or ABI changes that do not prevent a package from building correctly. Such recommendations can be expressed using RECOMMENDED: RECOMMENDED+= tiff>=3.6.1:../../graphics/tiff In addition to the above DEPENDS line, this denotes that while a package will build against tiff>=3.5.4, at least version 3.6.1 is recommended. RECOMMENDED entries will be turned into dependencies unless explicitly ignored (in which case a warning will be printed). Packages that are built with recommendations ignored may not be uploaded to ftp.NetBSD.org by developers and should not be used across different systems that may have different versions of binary packages installed. For security fixes, please update the package vulnerabilities file as well as setting RECOMMENDED, see for more information. If your package needs some executable to be able to run correctly and if there's agail no buildlink3.mk file, this is specified using the DEPENDS variable. The print/lyx package needs to be able to execute the latex binary from the teTeX package when it runs, and that is specified: DEPENDS+= teTeX-[0-9]*:../../print/teTeX The comment about wildcard dependencies from previous paragraph applies here, too. If your package needs files from another package to build, see the first part of the do-configure target print/ghostscript5 package (it relies on the jpeg sources being present in source form during the build): if [! -e ${_PKGSRCDIR}/graphics/jpeg/${WRKDIR:T}/jpeg-6b]; then \ cd ${_PKGSRCDIR}/../../graphics/jpeg && ${MAKE} extract; \ fi If you build any other packages that way, please make sure the working files are deleted too when this package's working files are cleaned up. The easiest way to do so is by adding a pre-clean target: pre-clean: cd ${_PKGSRCDIR}/../../graphics/jpeg && ${MAKE} clean Please also note the BUILD_USES_MSGFMT and BUILD_USES_GETTEXT_M4 definitions, which are provided as convenience definitions. The former works out whether msgfmt(1) is part of the base system, and, if it isn't, installs the devel/gettext package. The latter adds a build dependency on either an installed version of an older gettext package, or if it isn't, installs the devel/gettext-m4 package. Handling conflicts with other packages Your package may conflict with other packages a user might already have installed on his system, e.g. if your package installs the same set of files like another package in our pkgsrc tree. In this case you can set CONFLICTS to a space separated list of packages (including version string) your package conflicts with. For example x11/Xaw3d and x11/Xaw-Xpm install provide the same shared library, thus you set in pkgsrc/x11/Xaw3d/Makefile: CONFLICTS= Xaw-Xpm-[0-9]* and in pkgsrc/x11/Xaw-Xpm/Makefile: CONFLICTS= Xaw3d-[0-9]* Packages will automatically conflict with other packages with the name prefix and a different version string. Xaw3d-1.5 e.g. will automatically conflict with the older version Xaw3d-1.3. Packages that cannot or should not be built There are several reasons why a package might be instructed to not build under certain circumstances. If the package builds and runs on most platforms, the exceptions should be noted with NOT_FOR_PLATFORM. If the package builds and runs on a small handful of platforms, set ONLY_FOR_PLATFORM instead. If the package should be skipped (for example, because it provides functionality already provided by the system), set PKG_SKIP_REASON to a descriptive message. If the package should fail because some preconditions are not met, set PKG_FAIL_REASON to a descriptive message. IGNORE is deprecated because it didn't provide enough information to determine whether the build should fail. Packages which should not be deleted, once installedTo ensure that a package may not be deleted, once it has been installed, the PKG_PRESERVE definition should be set in the package Makefile. This will be carried into any binary package that is made from this pkgsrc entry. A preserved package will not be deleted using pkg_delete1 unless the -f option is used. Handling packages with security problems When a vulnerability is found, this should be noted in localsrc/security/advisories/pkg-vulnerabilities, and after the commit of that file, it should be copied to both /pub/NetBSD/packages/distfiles/pkg-vulnerabilities and /pub/NetBSD/packages/distfiles/vulnerabilities on ftp.NetBSD.org using localsrc/security/advisories/Makefile. In addition, if a buildlink3.mk file exists for an affected package, bumping PKGREVISION and creating a corresponding BUILDLINK_RECOMMENDED.pkg entry should be considered. See for more information about writing buildlink3.mk files and BUILDLINK_* definitions. Also, if the fix should be applied to the stable pkgsrc branch, be sure to submit a pullup request! How to handle compiler bugs Some source files trigger bugs in the compiler, based on combinations of compiler version and architecture and almost always relation to optimisation being enabled. Common symptoms are gcc internal errors or never finishing compiling a file. Typically a workaround involves testing the MACHINE_ARCH and compiler version, disabling optimisation for that file/MACHINE_ARCH/compiler combination, and documenting it in pkgsrc/doc/HACKS. See that file for a number of examples! How to handle incrementing versions when fixing an existing package When making fixes to an existing package it can be useful to change the version number in PKGNAME. To avoid conflicting with future versions by the original author, a nb1, nb2, ... suffix can be used on package versions by setting PKGREVISION=1 (2, ...). The nb is treated like a . by the pkg tools. e.g. DISTNAME= foo-17.42 PKGREVISION= 9 will result in a PKGNAME of foo-17.42nb9. When a new release of the package is released, the PKGREVISION should be removed. e.g. on a new minor release of the above package, things should be like: DISTNAME= foo-17.43Portability of packagesOne appealing feature of pkgsrc is that it runs on many different platforms. As a result, it is important to ensure, where possible, that packages in pkgsrc are portable. There are some particular details you should pay attention to while working on pkgsrc.${INSTALL}, ${INSTALL_DATA_DIR}, ...The BSD-compatible install supplied with some operating systems will not perform more than one operation at a time. As such, you should call ${INSTALL}, etc. like this:${INSTALL_DATA_DIR} ${PREFIX}/dir1 ${INSTALL_DATA_DIR} ${PREFIX}/dir2Possible downloading issuesPackages whose distfiles aren't available for plain downloading If you need to download from a dynamic URL you can set DYNAMIC_MASTER_SITES and a make fetch will call files/getsite.sh with the name of each file to download as an argument, expecting it to output the URL of the directory from which to download it. graphics/ns-cult3d is an example of this usage. If the download can't be automated, because the user must submit personal information to apply for a password, or must pay for the source, or whatever, you can set _FETCH_MESSAGE to a macro which displays a message explaining the situation. _FETCH_MESSAGE must be executable shell commands, not just a message. (Generally, it executes ${ECHO}). As of this writing, the following packages use this: audio/realplayer, cad/simian, devel/ipv6socket, emulators/vmware-module, fonts/acroread-jpnfont, sysutils/storage-manager, www/ap-aolserver, www/openacs. Try to be consistent with them. How to handle modified distfiles with the 'old' name Sometimes authors of a software package make some modifications after the software was released, and they put up a new distfile without changing the package's version number. If a package is already in pkgsrc at that time, the md5 checksum will no longer match. The correct way to work around this is to update the package's md5 checksum to match the package on the master site (beware, any mirrors may not be up to date yet!), and to remove the old distfile from ftp.NetBSD.org's /pub/NetBSD/packages/distfiles directory. Furthermore, a mail to the package's author seems appropriate making sure the distfile was really updated on purpose, and that no trojan horse or so crept in. Configuration gotchasShared libraries - libtoolpkgsrc supports many different machines, with different object formats like a.out and ELF, and varying abilities to do shared library and dynamic loading at all. To accompany this, varying commands and options have to be passed to the compiler, linker, etc. to get the Right Thing, which can be pretty annoying especially if you don't have all the machines at your hand to test things. The devel/libtool pkg can help here, as it just knows how to build both static and dynamic libraries from a set of source files, thus being platform independent.Here's how to use libtool in a pkg in seven simple steps:Add USE_LIBTOOL=yes to the package Makefile.For library objects, use ${LIBTOOL} --mode=compile ${CC} in place of ${CC}. You could even add it to the definition of CC, if only libraries are being built in a given Makefile. This one command will build both PIC and non-PIC library objects, so you need not have separate shared and non-shared library rules.For the linking of the library, remove any ar, ranlib, and ld -Bshareable commands, and instead use:${LIBTOOL} --mode=link ${CC} -o ${.TARGET:.a=.la} ${OBJS:.o=.lo} -rpath ${PREFIX}/lib -version-info major:minorNote that the library is changed to have a .la extension, and the objects are changed to have a .lo extension. Change OBJS as necessary. This automatically creates all of the .a, .so.major.minor, and ELF symlinks (if necessary) in the build directory. Be sure to include -version-info, especially when major and minor are zero, as libtool will otherwise strip off the shared library version. From the libtool manual: So, libtool library versions are described by three integers: CURRENT The most recent interface number that this library implements. REVISION The implementation number of the CURRENT interface. AGE The difference between the newest and oldest interfaces that this library implements. In other words, the library implements all the interface numbers in the range from number 'CURRENT - AGE' to 'CURRENT'. If two libraries have identical CURRENT and AGE numbers, then the dynamic linker chooses the library with the greater REVISION number. The -release option will produce different results for a.out and ELF (excluding symlinks) in only one case. An ELF library of the form libfoo-release.so.x.y will have a symlink of libfoo.so.x.y on an a.out platform. This is handled automatically.The -rpath argument is the install directory of the library being built.In the PLIST, include all of the .a, .la, and .so, .so.major and .so.major.minor files.When linking shared object (.so) files, i.e. files that are loaded via dlopen(3), NOT shared libraries, use -module -avoid-version to prevent them getting version tacked on.The PLIST file gets the foo.so entry.When linking programs that depend on these libraries before they are installed, preface the cc1 or ld1 line with ${LIBTOOL} --mode=link, and it will find the correct libraries (static or shared), but please be aware that libtool will not allow you to specify a relative path in -L (such as -L../somelib), because it expects you to change that argument to be the .la file. e.g.${LIBTOOL} --mode=link ${CC} -o someprog -L../somelib -lsomelibshould be changed to:${LIBTOOL} --mode=link ${CC} -o someprog ../somelib/somelib.laand it will do the right thing with the libraries.When installing libraries, preface the install1 or cp1 command with ${LIBTOOL} --mode=install, and change the library name to .la. e.g.${LIBTOOL} --mode=install ${BSD_INSTALL_DATA} ${SOMELIB:.a=.la} ${PREFIX}/libThis will install the static .a, shared library, any needed symlinks, and run ldconfig8. In your PLIST, include all of the .a, .la, and .so, .so.CURRENT and .so.CURRENT.REVISION files (this is a change from the previous behaviour). Using libtool on GNU packages that already support libtoolAdd USE_LIBTOOL=yes to the package Makefile. This will override the package's own libtool in most cases. For older libtool using packages, libtool is made by ltconfig script during the do-configure step; you can check the libtool script location by doing make configure; find work*/ -name libtool. LIBTOOL_OVERRIDE specifies which libtool scripts, relative to WRKSRC, to override. By default, it is set to libtool */libtool */*/libtool. If this does not match the location of the package's libtool script(s), set it as appropriate. If you do not need *.a static libraries built and installed, then use SHLIBTOOL_OVERRIDE instead. If your package makes use of the platform independent library for loading dynamic shared objects, that comes with libtool (libltdl), you should include the libtool buildlink3.mk (and set USE_BUILDLINK3=YES).Some packages use libtool incorrectly so that the package may not work or build in some circumstances. Some of the more common errors are:The inclusion of a shared object (-module) as a dependent library in an executable or library. This in itself isn't a problem if one of two things has been done:The shared object is named correctly, i.e. libfoo.la, not foo.laThe -dlopen option is used when linking an executable.The use of libltdl without the correct calls to initialisation routines. The function lt_dlinit() should be called and the macro LTDL_SET_PRELOADED_SYMBOLS included in executables.GNU Autoconf/AutomakeIf a package needs GNU autoconf or automake to be executed to regenerate the configure script and Makefile.in makefile templates, then they should be executed in a pre-configure target. Two Makefile fragments are provided in pkgsrc/mk/autoconf.mk and pkgsrc/mk/automake.mk to help dealing with these tools. See comments in these files for details. For packages that need only autoconf: AUTOCONF_REQD= 2.50 # if default version is not good enough ... pre-configure: cd ${WRKSRC}; ${AUTOCONF} ... .include "../../mk/autoconf.mk" and for packages that need automake and autoconf: AUTOMAKE_REQD= 1.7.1 # if default version is not good enough ... pre-configure: cd ${WRKSRC}; \ ${ACLOCAL}; \ ${AUTOHEADER}; \ ${AUTOMAKE} -a --foreign -i; \ ${AUTOCONF} ... .include "../mk/automake.mk" Packages which use GNU Automake will almost certainly require GNU Make, but that's automatically provided for you in mk/automake.mk. There are times when the configure process makes additional changes to the generated files, which then causes the build process to try to re-execute the automake sequence. This is prevented by touching various files in the configure stage. If this causes problems with your package you can set AUTOMAKE_OVERRIDE=NO in the package Makefile. Building considerationsCPP defines To port an application to NetBSD, it's usually necessary for the compiler to be able to judge the system on which it's compiling, and we use definitions so that the C pre-processor can do this. To test whether you are working on a 4.4 BSD-derived system, you should use the BSD definition, which is defined in <sys/param.h> on said systems. #include <sys/param.h>and then you can surround the BSD-specific parts of your package's C/C++ code using this conditional:#if (defined(BSD) && BSD >= 199306) ... #endifPlease use the __NetBSD__ definition sparingly - it should only apply to features of NetBSD that are not present in other 4.4-lite derived BSDs.Package specific actionsPackage configuration files Packages should be taught to look for their configuration files in ${PKG_SYSCONFDIR}, which is passed through to the configure and build processes. PKG_SYSCONFDIR may be customized in various ways by setting other make variables: PKG_SYSCONFBASE is the main config directory under which all package configuration files are to be found. This defaults to ${PREFIX}/etc, but may be overridden in /etc/mk.conf. PKG_SYSCONFSUBDIR is the subdirectory of PKG_SYSCONFBASE under which the configuration files for a particular package may be found, e.g. the Apache configuration files may all be found under the httpd/ subdirectory of ${PKG_SYSCONFBASE}. This should be set in the package Makefile. By default, PKG_SYSCONFDIR is set to ${PKG_SYSCONFBASE}/${PKG_SYSCONFSUBDIR}, but this may be overridden by setting PKG_SYSCONFDIR.${PKG_SYSCONFVAR} for a particular package, where PKG_SYSCONFVAR defaults to ${PKGBASE}. This is not meant to be set by a package Makefile, but is reserved for users who wish to override the PKG_SYSCONFDIR setting for a particular package with a special location. The only variables that users should customize are PKG_SYSCONFBASE and PKG_SYSCONFDIR.${PKG_SYSCONFVAR}. Users will typically want to set PKG_SYSCONFBASE to /etc, or to accept the default location of ${PREFIX}/etc. User interactionOccasionally, packages require interaction from the user, and this can be in a number of ways:help in fetching the distfileshelp to configure the package before it is builthelp during the build processhelp during the installation of a packageThe INTERACTIVE_STAGE definition is provided to notify the pkgsrc mechanism of an interactive stage which will be needed, and this should be set in the package's Makefile. e.g.INTERACTIVE_STAGE= buildMultiple interactive stages can be specified:INTERACTIVE_STAGE= configure installHandling licensesA package may underly a license which the user has or has not agreed to accept. Usually, packages that underly well-known Open Source licenses (e.g. the GNU Public License, GPL) won't have any special license tags added in pkgsrc which require special action by the user of such packages, but there are quite a number of other licenses out there that pkgsrc users may not be able to follow, for whatever reasons. For these cases, pkgsrc contains a mechanism to note that a package underlies a certain license, and the user has to accept the license before the package can be installed. Placing a certain package under a certain license works by setting the LICENSE variable to a string identifying the license, e.g. in graphics/graphviz: LICENSE= graphviz-license When trying to build, the user will get a notice that the package underlies a license which he hasn't accepted (yet): % make ===> graphviz-1.12 has an unacceptable license: graphviz-license. ===> To build this package, add this line to your /etc/mk.conf: ===> ACCEPTABLE_LICENSES+=graphviz-license ===> To view the license, enter "/usr/bin/make show-license". *** Error code 1 The license can be viewed with make show-license, and if it is considered appropriate, the line printed above can be added to /etc/mk.conf to indicate acceptance of the particular license:ACCEPTABLE_LICENSES+=graphviz-licenseWhen adding a package with a new license, the license text should be added to pkgsrc/licenses for displaying. A list of known licenses can be seen in this directory as well as by looking at the list of (commented out) ACCEPTABLE_LICENSES variable settings in pkgsrc/mk/defaults/mk.conf. Is there is a really pressing need to accept all licenses at once, like when trying to download or mirror all distfiles or doing a bulk build to test if all packages in pkgsrc build, this can be done by setting _ACCEPTABLE=yes. Creating an account from a package There are two make variables used to control the creation of package-specific groups and users at pre-install time. The first is PKG_GROUPS, which is a list of group[:groupid] elements, where the groupid is optional. The second is PKG_USERS, which is a list of elements of the form: user:group[:[userid][:[description][:[home][:shell]]]] where only the user and group are required, the rest being optional. A simple example is: PKG_GROUPS= foogroup PKG_USERS= foouser:foogroup A more complex example is that creates two groups and two users is: PKG_GROUPS= group1 group2:1005 PKG_USERS= first:group1::First\\ User \ second:group2::Second\\ User:/home/second:${SH} By default, a new user will have home directory /nonexistent, and login shell /sbin/nologin unless they are specified as part of the user element. The package Makefile must also set USE_PKGINSTALL=YES. This will cause the users and groups to be created at pre-install time, and the admin will be prompted to remove them at post-deinstall time. Automatic creation of the users and groups can be toggled on and off by setting the PKG_CREATE_USERGROUP variable prior to package installation. Installing score files Certain packages, most of them in the games category, install a score file that allows all users on the system to record their highscores. In order for this to work, the binaries need to be installed setgid and the score files owned by the appropriate group and/or owner (traditionally the "games" user/group). The following variables, documented in more detail in mk/defaults/mk.conf, control this behaviour: SETGIDGAME, GAMEDATAMODE, GAMEGRP, GAMEMODE, GAMEOWN. Note that per default, setgid installation of games is disabled; setting SETGIDGAME=YES will set all the other variables accordingly. A package should therefor never hard code file ownership or access permissions but rely on INSTALL_GAME and INSTALL_GAME_DATA to set these correctly. Packages providing login shells If the purpose of the package is to provide a login shell, the variable PKG_SHELL should contain the full pathname of the shell executable installed by this package. The package Makefile must also set USE_PKGINSTALL=YES to use the automatically generated INSTALL/DEINSTALL scripts. An example taken from shells/zsh: USE_PKGINSTALL= YES PKG_SHELL= ${PREFIX}/bin/zsh The shell is registered into /etc/shells file automatically in the post-install target by the generated INSTALL script and removed in the deinstall target by the DEINSTALL script. Packages containing perl scripts If your package contains interpreted perl scripts, set REPLACE_PERL to ensure that the proper interpreter path is set. REPLACE_PERL should contain a list of scripts, relative to WRKSRC, that you want adjusted. Packages with hardcoded paths to other interpreters Your package may also contain scripts with hardcoded paths to other interpreters besides (or as well as) perl. To correct the full pathname to the script interpreter, you need to set the following definitions in your Makefile (we shall use tclsh in this example): REPLACE_INTERPRETER+= tcl _REPLACE.tcl.old= .*/bin/tclsh _REPLACE.tcl.new= ${PREFIX}/bin/tclsh _REPLACE_FILES.tcl= ...list of tcl scripts which need to be fixed, relative to ${WRKSRC}, just as in REPLACE_PERLPackages installing perl modulesMakefiles of packages providing perl5 modules should include the Makefile fragment ../../lang/perl5/module.mk. It provides a do-configure target for the standard perl configuration for such modules as well as various hooks to tune this configuration. See comments in this file for details.Perl5 modules will install into different places depending on the version of perl used during the build process. To address this, pkgsrc will append lines to the PLIST corresponding to the files listed in the installed .packlist file generated by most perl5 modules. This is invoked by defining PERL5_PACKLIST to a space-separated list of paths to packlist files, e.g.:PERL5_PACKLIST= ${PERL5_SITEARCH}/auto/Pg/.packlistThe variables PERL5_SITELIB, PERL5_SITEARCH, and PERL5_ARCHLIB represent the three locations in which perl5 modules may be installed, and may be used by perl5 packages that don't have a packlist. These three variables are also substituted for in the PLIST.Packages installing info files Some packages install info files or use the makeinfo or install-info commands. Each of the info files: is considered to be installed in the directory ${PREFIX}/${INFO_DIR},is registered in the Info directory file ${PREFIX}/${INFO_DIR}/dir,and must be listed as a filename in the INFO_FILES variable in the package Makefile. INFO_DIR defaults to info and can be overridden in the package Makefile. INSTALL and DEINSTALL scripts will be generated to handle registration of the info files in the Info directory file. The install-info command used for the info files registration is either provided by the system, or by a special purpose package automatically added as dependency if needed. A package which needs the makeinfo command at build time must define the variable USE_MAKEINFO in its Makefile. If a minimum version of the makeinfo command is needed it should be noted with the TEXINFO_REQD variable in the package Makefile. By default, a minimum version of 3.12 is required. If the system does not provide a makeinfo command or if it does not match the required minimum, a build dependency on the devel/gtexinfo package will be added automatically. The build and installation process of the software provided by the package should not use the install-info command as the registration of info files is the task of the package INSTALL script, and it must use the appropriate makeinfo command. To achieve this goal the pkgsrc infrastructure creates overriding scripts for the install-info and makeinfo commands in a directory listed early in PATH. The script overriding install-info has no effect except the logging of a message. The script overriding makeinfo logs a message and according to the value of USE_MAKEINFO and TEXINFO_REQD either run the appropriate makeinfo command or exit on error. Packages installing GConf2 data files If a package installs .schemas or .entries files, used by GConf2, you need to take some extra steps to make sure they get registered in the database: Include ../../devel/GConf2/schemas.mk instead of its buildlink3.mk file. This takes care of rebuilding the GConf2 database at installation and deinstallation time, and tells the package where to install GConf2 data files using some standard configure arguments. It also disallows any access to the database directly from the package. Ensure that the package installs its .schemas files under ${PREFIX}/share/gconf/schemas. If they get installed under ${PREFIX}/etc, you will need to manually patch the package. Check the PLIST and remove any entries under the etc/gconf directory, as they will be handled automatically. See for more information. Define the GCONF2_SCHEMAS variable in your Makefile with a list of all .schemas files installed by the package, if any. Names must not contain any directories in them. Define the GCONF2_ENTRIES variable in your Makefile with a list of all .entries files installed by the package, if any. Names must not contain any directories in them. Packages installing scrollkeeper data files If a package installs .omf files, used by scrollkeeper, you need to take some extra steps to make sure they get registered in the database: Include ../../textproc/scrollkeeper/omf.mk instead of its buildlink3.mk file. This takes care of rebuilding the scrollkeeper database at installation and deinstallation time, and disallows any access to it directly from the package. Check the PLIST and remove any entries under the libdata/scrollkeeper directory, as they will be handled automatically. Remove the share/omf directory from the PLIST. It will be handled by scrollkeeper. Packages installing X11 fonts If a package installs font files, you will need to rebuild the fonts database in the directory where they get installed at installation and deinstallation time. This can be automatically done by using mk/fonts.mk, which you need to include in your Makefile. When the file is included, you can list the directories where fonts are installed in the FONTS_type_DIRS variables, where type can be one of TTF, TYPE1 or X11. Also make sure that the database file fonts.dir is not listed in the PLIST. Note that you should not create new directories for fonts; instead use the standard ones to avoid that the user needs to manually configure his X server to find them. Packages installing GTK2 modulesIf a package installs gtk2 immodules or loaders, you need to take some extra steps to get them registered in the GTK2 database properly: Include ../../x11/gtk2/modules.mk instead of its buildlink3.mk file. This takes care of rebuilding the database at installation and deinstallation time. Set GTK2_IMMODULES=YES if your package installs GTK2 immodules. Set GTK2_LOADERS=YES if your package installs GTK2 loaders. Patch the package to not touch any of the gtk2 databases directly. These are: libdata/gtk-2.0/gdk-pixbuf.loaderslibdata/gtk-2.0/gtk.immodules Check the PLIST and remove any entries under the libdata/gtk-2.0 directory, as they will be handled automatically. Packages installing SGML or XML data If a package installs SGML or XML data files that need to be registered in system-wide catalogs (like DTDs, sub-catalogs, etc.), you need to take some extra steps: Include ../../textproc/xmlcatmgr/catalogs.mk in your Makefile, which takes care of registering those files in system-wide catalogs at installation and deinstallation time. Set SGML_CATALOGS to the full path of any SGML catalogs installed by the package. Set XML_CATALOGS to the full path of any XML catalogs installed by the package. Set SGML_ENTRIES to individual entries to be added to the SGML catalog. These come in groups of three strings; see xmlcatmgr(1) for more information (specifically, arguments recognized by the 'add' action). Note that you will normally not use this variable. Set XML_ENTRIES to individual entries to be added to the XML catalog. These come in groups of three strings; see xmlcatmgr(1) for more information (specifically, arguments recognized by the 'add' action). Note that you will normally not use this variable. Packages installing extensions to the MIME database If a package provides extensions to the MIME database by installing .xml files inside ${PREFIX}/share/mime/packages, you need to take some extra steps to ensure that the database is kept consistent with respect to these new files: Include ../../databases/shared-mime-info/mimedb.mk (avoid using the buildlink3.mk file from this same directory, which is reserved for inclusion from other buildlink3.mk files). It takes care of rebuilding the MIME database at installation and deinstallation time, and disallows any access to it directly from the package. Check the PLIST and remove any entries under the share/mime directory, except for files saved under share/mime/packages. The former are handled automatically by the update-mime-database program, but the later are package-dependent and must be removed by the package that installed them in the first place. Remove any share/mime/* directories from the PLIST. They will be handled by the shared-mime-info package. Packages using intltool If a package uses intltool during its build, include the ../../textproc/intltool/buildlink3.mk file, which forces it to use the intltool package provided by pkgsrc, instead of the one bundled with the distribution file. This tracks intltool's build-time dependencies and uses the latest available version; this way, the package benefits of any bug fixes that may have appeared since it was released. Feedback to the author If you have found any bugs in the package you make available, if you had to do special steps to make it run under NetBSD or if you enhanced the software in various other ways, be sure to report these changes back to the original author of the program! With that kind of support, the next release of the program can incorporate these fixes, and people not using the NetBSD packages system can win from your efforts. Support the idea of free software! DebuggingTo check out all the gotchas when building a package, here are the steps that I do in order to get a package working. Please note this is basically the same as what was explained in the previous sections, only with some debugging aids.Be sure to set PKG_DEVELOPER=1 in /etc/mk.confInstall pkgtools/url2pkg, create a directory for a new package, change into it, then run url2pkg:% mkdir /usr/pkgsrc/category/examplepkg % cd /usr/pkgsrc/category/examplepkg % url2pkg http://www.example.com/path/to/distfile.tar.gzEdit the Makefile as requested.Fill in the DESCR fileRun make configure Add any dependencies glimpsed from documentation and the configure step to the package's Makefile.Make the package compile, doing multiple rounds of% make % pkgvi ${WRKSRC}/some/file/that/does/not/compile % mkpatches % patchdiff % mv ${WRKDIR}/.newpatches/* patches % make mps % make cleanDoing as non-root user will ensure that no files are modified that shouldn't be, especially during the build phase. mkpatches, patchdiff and pkgvi are from the pkgtools/pkgdiff package. Look at the Makefile, fix if necessary; see .Generate a PLIST:# make install # make print-PLIST >PLIST # make deinstall # make install # make deinstallYou usually need to be root to do this. Look if there are any files left:# make print-PLISTIf this reveals any files that are missing in PLIST, add them.Now that the PLIST is OK, install the package again and make a binary package:# make reinstall # make packageDelete the installed package:# pkg_delete blubRepeat the above make print-PLIST command, which shouldn't find anything now:# make print-PLISTReinstall the binary package:# pkgadd .../blub.tgzPlay with it. Make sure everything works.Run pkglint from pkgtools/pkglint, and fix the problems it reports:# pkglintSubmit (or commit, if you have cvs access); see .Submitting and CommittingSubmitting your packages You have to separate between binary and normal (source) packages here: precompiled binary packages Our policy is that we accept binaries only from pkgsrc developers to guarantee that the packages don't contain any trojan horses etc. This is not to piss anyone off but rather to protect our users! You're still free to put up your home-made binary packages and tell the world where to get them. packages First, check that your package is complete, compiles and runs well; see and the rest of this document. Next, generate an uuencoded gzipped tar(1) archive, preferably with all files in a single directory. Finally, send-pr with category pkg, a synopsis which includes the package name and version number, a short description of your package (contents of the COMMENT variable or DESCR file are OK) and attach the archive to your PR. If you want to submit several packages, please send a separate PR for each one, it's easier for us to track things that way. Alternatively, you can also import new packages into pkgsrc-wip (pkgsrc work-in-progress); see the homepage at for details. Committing: Importing a package into CVS This section is only of interest for pkgsrc developers with write access to the pkgsrc repository. Please remember that cvs imports files relative to the current working directory, and that the pathname that you give the cvs import command is so that it knows where to place the files in the repository. Newly created packages should be imported with a vendor tag of TNF and a release tag of pkgsrc-base, e.g: % cd .../pkgsrc/category/pkgname % cvs import pkgsrc/category/pkgname TNF pkgsrc-base Remember to move the directory from which you imported out of the way, or cvs will complain the next time you cvs update your source tree. Also don't forget to add the new package to the category's Makefile. The commit message of the initial import should include part of the DESCR file, so people reading the mailing lists know what the package is/does. Please note all package updates/additions in pkgsrc/doc/CHANGES. It's very important to keep this file up to date and conforming to the existing format, because it will be used by scripts to automatically update pages on www.NetBSD.org and other sites. Additionally, check the pkgsrc/doc/TODO file and remove the entry for the package you updated, in case it was mentioned there. For new packages, cvs import is preferred to cvs add because the former gets everything with a single command, and provides a consistent tag. Updating a package to a newer version Please always put a concise, appropriate and relevant summary of the changes between old and new versions into the commit log when updating a package. There are various reasons for this: A URL is volatile, and can change over time. It may go away completely or its information may be overwritten by newer information. Having the change information between old and new versions in our CVS repository is very useful for people who use either cvs or anoncvs. Having the change information between old and new versions in our CVS repository is very useful for people who read the pkgsrc-changes mailing list, so that they can make tactical decisions about when to upgrade the package. Please also recognise that, just because a new version of a package has been released, it should not automatically be upgraded in the CVS repository. We prefer to be conservative in the packages that are included in pkgsrc - development or beta packages are not really the best thing for most places in which pkgsrc is used. Please use your judgement about what should go into pkgsrc, and bear in mind that stability is to be preferred above new and possibly untested features. Moving a package in pkgsrcMake a copy of the directory somewhere else.Remove all CVS dirs. Alternatively to the first two steps you can also do: % cvs -d user@cvs.NetBSD.org:/cvsroot export -D today pkgsrc/category/package and use that for further work. Fix CATEGORIES and any DEPENDS paths that just did ../package instead of ../../category/package.cvs import the modified package in the new place.Check if any package depends on it: % cd /usr/pkgsrc % grep /package */*/Makefile* */*/buildlink* Fix paths in packages from step 5 to point to new location.cvs rm (-f) the package at the old location.Remove from oldcategory/Makefile.Add to newcategory/Makefile.Commit the changed and removed files:% cvs commit oldcategory/package oldcategory/Makefile newcategory/Makefile (and any packages from step 5, of course). A simple example package: bisonWe checked to find a piece of software that wasn't in the packages collection, and picked GNU bison. Quite why someone would want to have bison when Berkeley yacc is already present in the tree is beyond us, but it's useful for the purposes of this exercise.filesMakefile# $NetBSD$ # DISTNAME= bison-1.25 CATEGORIES= devel MASTER_SITES= ${MASTER_SITE_GNU} MAINTAINER= thorpej@NetBSD.org HOMEPAGE= http://www.gnu.org/software/bison/bison.html COMMENT= GNU yacc clone GNU_CONFIGURE= yes INFO_FILES= bison.info .include "../../mk/bsd.pkg.mk"DESCRGNU version of yacc. Can make re-entrant parsers, and numerous other improvements. Why you would want this when Berkeley yacc1 is part of the NetBSD source tree is beyond me.PLIST@comment $NetBSD$ bin/bison man/man1/bison.1.gz share/bison.simple share/bison.hairyChecking a package with pkglintThe NetBSD package system comes with pkgtools/pkglint which helps to check the contents of these files. After installation it is quite easy to use, just change to the directory of the package you wish to examine and execute pkglint:$ pkglint OK: checking ./DESCR. OK: checking Makefile. OK: checking distinfo. OK: checking patches/patch-aa. looks fine.Depending on the supplied command line arguments (see pkglint(1)) more verbose checks will be performed. Use e.g. pkglint -v for a very verbose check.Steps for building, installing, packagingCreate the directory where the package lives, plus any auxiliary directories:# cd /usr/pkgsrc/lang # mkdir bison # cd bison # mkdir patchesCreate Makefile, DESCR and PLIST (see) then continue with fetching the distfile:# make fetch >> bison-1.25.tar.gz doesn't seem to exist on this system. >> Attempting to fetch from ftp://prep.ai.mit.edu/pub/gnu//. Requesting ftp://prep.ai.mit.edu/pub/gnu//bison-1.25.tar.gz (via ftp://orpheus.amdahl.com:80/) ftp: Error retrieving file: 500 Internal error >> Attempting to fetch from ftp://wuarchive.wustl.edu/systems/gnu//. Requesting ftp://wuarchive.wustl.edu/systems/gnu//bison-1.25.tar.gz (via ftp://orpheus.amdahl.com:80/) ftp: Error retrieving file: 500 Internal error >> Attempting to fetch from ftp://ftp.freebsd.org/pub/FreeBSD/distfiles//. Requesting ftp://ftp.freebsd.org/pub/FreeBSD/distfiles//bison-1.25.tar.gz (via ftp://orpheus.amdahl.com:80/) Successfully retrieved file.Generate the checksum of the distfile into distinfo:# make makesumNow compile:# make >> Checksum OK for bison-1.25.tar.gz. ===> Extracting for bison-1.25 ===> Patching for bison-1.25 ===> Ignoring empty patch directory ===> Configuring for bison-1.25 creating cache ./config.cache checking for gcc... cc checking whether we are using GNU C... yes checking for a BSD compatible install... /usr/bin/install -c -o bin -g bin checking how to run the C preprocessor... cc -E checking for minix/config.h... no checking for POSIXized ISC... no checking whether cross-compiling... no checking for ANSI C header files... yes checking for string.h... yes checking for stdlib.h... yes checking for memory.h... yes checking for working const... yes checking for working alloca.h... no checking for alloca... yes checking for strerror... yes updating cache ./config.cache creating ./config.status creating Makefile ===> Building for bison-1.25 cc -c -DSTDC_HEADERS=1 -DHAVE_STRING_H=1 -DHAVE_STDLIB_H=1 -DHAVE_MEMORY_H=1 -DHAVE_ALLOCA=1 -DHAVE_STRERROR=1 -I./../include -g LR0.c cc -c -DSTDC_HEADERS=1 -DHAVE_STRING_H=1 -DHAVE_STDLIB_H=1 -DHAVE_MEMORY_H=1 -DHAVE_ALLOCA=1 -DHAVE_STRERROR=1 -I./../include -g allocate.c cc -c -DSTDC_HEADERS=1 -DHAVE_STRING_H=1 -DHAVE_STDLIB_H=1 -DHAVE_MEMORY_H=1 -DHAVE_ALLOCA=1 -DHAVE_STRERROR=1 -I./../include -g closure.c cc -c -DSTDC_HEADERS=1 -DHAVE_STRING_H=1 -DHAVE_STDLIB_H=1 -DHAVE_MEMORY_H=1 -DHAVE_ALLOCA=1 -DHAVE_STRERROR=1 -I./../include -g conflicts.c cc -c -DSTDC_HEADERS=1 -DHAVE_STRING_H=1 -DHAVE_STDLIB_H=1 -DHAVE_MEMORY_H=1 -DHAVE_ALLOCA=1 -DHAVE_STRERROR=1 -I./../include -g derives.c cc -c -DXPFILE=\"/usr/pkg/share/bison.simple\" -DXPFILE1=\"/usr/pkg/share/bison.hairy\" -DSTDC_HEADERS=1 -DHAVE_STRING_H=1 -DHAVE_STDLIB_H=1 -DHAVE_MEMORY_H=1 -DHAVE_ALLOCA=1 -DHAVE_STRERROR=1 -g ./files.c cc -c -DSTDC_HEADERS=1 -DHAVE_STRING_H=1 -DHAVE_STDLIB_H=1 -DHAVE_MEMORY_H=1 -DHAVE_ALLOCA=1 -DHAVE_STRERROR=1 -I./../include -g getargs.c cc -c -DSTDC_HEADERS=1 -DHAVE_STRING_H=1 -DHAVE_STDLIB_H=1 -DHAVE_MEMORY_H=1 -DHAVE_ALLOCA=1 -DHAVE_STRERROR=1 -I./../include -g gram.c cc -c -DSTDC_HEADERS=1 -DHAVE_STRING_H=1 -DHAVE_STDLIB_H=1 -DHAVE_MEMORY_H=1 -DHAVE_ALLOCA=1 -DHAVE_STRERROR=1 -I./../include -g lalr.c cc -c -DSTDC_HEADERS=1 -DHAVE_STRING_H=1 -DHAVE_STDLIB_H=1 -DHAVE_MEMORY_H=1 -DHAVE_ALLOCA=1 -DHAVE_STRERROR=1 -I./../include -g lex.c cc -c -DSTDC_HEADERS=1 -DHAVE_STRING_H=1 -DHAVE_STDLIB_H=1 -DHAVE_MEMORY_H=1 -DHAVE_ALLOCA=1 -DHAVE_STRERROR=1 -I./../include -g main.c cc -c -DSTDC_HEADERS=1 -DHAVE_STRING_H=1 -DHAVE_STDLIB_H=1 -DHAVE_MEMORY_H=1 -DHAVE_ALLOCA=1 -DHAVE_STRERROR=1 -I./../include -g nullable.c cc -c -DSTDC_HEADERS=1 -DHAVE_STRING_H=1 -DHAVE_STDLIB_H=1 -DHAVE_MEMORY_H=1 -DHAVE_ALLOCA=1 -DHAVE_STRERROR=1 -I./../include -g output.c cc -c -DSTDC_HEADERS=1 -DHAVE_STRING_H=1 -DHAVE_STDLIB_H=1 -DHAVE_MEMORY_H=1 -DHAVE_ALLOCA=1 -DHAVE_STRERROR=1 -I./../include -g print.c cc -c -DSTDC_HEADERS=1 -DHAVE_STRING_H=1 -DHAVE_STDLIB_H=1 -DHAVE_MEMORY_H=1 -DHAVE_ALLOCA=1 -DHAVE_STRERROR=1 -I./../include -g reader.c cc -c -DSTDC_HEADERS=1 -DHAVE_STRING_H=1 -DHAVE_STDLIB_H=1 -DHAVE_MEMORY_H=1 -DHAVE_ALLOCA=1 -DHAVE_STRERROR=1 -I./../include -g reduce.c cc -c -DSTDC_HEADERS=1 -DHAVE_STRING_H=1 -DHAVE_STDLIB_H=1 -DHAVE_MEMORY_H=1 -DHAVE_ALLOCA=1 -DHAVE_STRERROR=1 -I./../include -g symtab.c cc -c -DSTDC_HEADERS=1 -DHAVE_STRING_H=1 -DHAVE_STDLIB_H=1 -DHAVE_MEMORY_H=1 -DHAVE_ALLOCA=1 -DHAVE_STRERROR=1 -I./../include -g warshall.c cc -c -DSTDC_HEADERS=1 -DHAVE_STRING_H=1 -DHAVE_STDLIB_H=1 -DHAVE_MEMORY_H=1 -DHAVE_ALLOCA=1 -DHAVE_STRERROR=1 -I./../include -g version.c cc -c -DSTDC_HEADERS=1 -DHAVE_STRING_H=1 -DHAVE_STDLIB_H=1 -DHAVE_MEMORY_H=1 -DHAVE_ALLOCA=1 -DHAVE_STRERROR=1 -I./../include -g getopt.c cc -c -DSTDC_HEADERS=1 -DHAVE_STRING_H=1 -DHAVE_STDLIB_H=1 -DHAVE_MEMORY_H=1 -DHAVE_ALLOCA=1 -DHAVE_STRERROR=1 -I./../include -g getopt1.c cc -g -o bison LR0.o allocate.o closure.o conflicts.o derives.o files.o getargs.o gram.o lalr.o lex.o main.o nullable.o output.o print.o reader.o reduce.o symtab.o warshall.o version.o getopt.o getopt1.o ./files.c:240: warning: mktemp() possibly used unsafely, consider using mkstemp() rm -f bison.s1 sed -e "/^#line/ s|bison|/usr/pkg/share/bison|" < ./bison.simple > bison.s1Everything seems OK, so install the files:# make install >> Checksum OK for bison-1.25.tar.gz. ===> Installing for bison-1.25 sh ./mkinstalldirs /usr/pkg/bin /usr/pkg/share /usr/pkg/info /usr/pkg/man/man1 rm -f /usr/pkg/bin/bison cd /usr/pkg/share; rm -f bison.simple bison.hairy rm -f /usr/pkg/man/man1/bison.1 /usr/pkg/info/bison.info* install -c -o bin -g bin -m 555 bison /usr/pkg/bin/bison /usr/bin/install -c -o bin -g bin -m 644 bison.s1 /usr/pkg/share/bison.simple /usr/bin/install -c -o bin -g bin -m 644 ./bison.hairy /usr/pkg/share/bison.hairy cd .; for f in bison.info*; do /usr/bin/install -c -o bin -g bin -m 644 $f /usr/pkg/info/$f; done /usr/bin/install -c -o bin -g bin -m 644 ./bison.1 /usr/pkg/man/man1/bison.1 ===> Registering installation for bison-1.25You can now use bison, and also - if you decide so - remove it with pkg_delete bison. Should you decide that you want a binary package, do this now:# make package >> Checksum OK for bison-1.25.tar.gz. ===> Building package for bison-1.25 Creating package bison-1.25.tgz Registering depends:. Creating gzip'd tar ball in '/u/pkgsrc/lang/bison/bison-1.25.tgz'Now that you don't need the source and object files any more, clean up:# make clean ===> Cleaning for bison-1.25Build logsBuilding figlet# make ===> Checking for vulnerabilities in figlet-2.2.1nb2 => figlet221.tar.gz doesn't seem to exist on this system. => Attempting to fetch figlet221.tar.gz from ftp://ftp.figlet.org/pub/figlet/program/unix/. => [172219 bytes] Connected to ftp.plig.net. 220 ftp.plig.org NcFTPd Server (licensed copy) ready. 331 Guest login ok, send your complete e-mail address as password. 230-You are user #5 of 500 simultaneous users allowed. 230- 230- ___ _ _ _ 230- | _| |_ ___ ___| |_|___ ___ ___ ___ 230- | _| _| . |_| . | | | . |_| . | _| . | 230- |_| |_| | _|_| _|_|_|_ |_|___|_| |_ | 230- |_| |_| |___| |___| 230- 230-** Welcome to ftp.plig.org ** 230- 230-Please note that all transfers from this FTP site are logged. If you 230-do not like this, please disconnect now. 230- 230-This arhive is available via 230- 230-HTTP: http://ftp.plig.org/ 230-FTP: ftp://ftp.plig.org/ (max 500 connections) 230-RSYNC: rsync://ftp.plig.org/ (max 30 connections) 230- 230-Please email comments, bug reports and requests for packages to be 230-mirrored to ftp-admin@plig.org. 230- 230- 230 Logged in anonymously. Remote system type is UNIX. Using binary mode to transfer files. 200 Type okay. 250 "/pub" is new cwd. 250-"/pub/figlet" is new cwd. 250- 250-Welcome to the figlet archive at ftp.figlet.org 250- 250- ftp://ftp.figlet.org/pub/figlet/ 250- 250-The official FIGlet web page is: 250- http://www.figlet.org/ 250- 250-If you have questions, please mailto:info@figlet.org. If you want to 250-contribute a font or something else, you can email us. 250 250 "/pub/figlet/program" is new cwd. 250 "/pub/figlet/program/unix" is new cwd. local: figlet221.tar.gz remote: figlet221.tar.gz 502 Unimplemented command. 227 Entering Passive Mode (195,40,6,41,246,104) 150 Data connection accepted from 84.128.86.72:65131; transfer starting for figlet221.tar.gz (172219 bytes). 38% |************** | 65800 64.16 KB/s 00:01 ETA 226 Transfer completed. 172219 bytes received in 00:02 (75.99 KB/s) 221 Goodbye. => Checksum OK for figlet221.tar.gz. ===> Extracting for figlet-2.2.1nb2 ===> Required installed package ccache-[0-9]*: ccache-2.3nb1 found ===> Patching for figlet-2.2.1nb2 ===> Applying pkgsrc patches for figlet-2.2.1nb2 ===> Overriding tools for figlet-2.2.1nb2 ===> Creating toolchain wrappers for figlet-2.2.1nb2 ===> Configuring for figlet-2.2.1nb2 ===> Building for figlet-2.2.1nb2 gcc -O2 -DDEFAULTFONTDIR=\"/usr/pkg/share/figlet\" -DDEFAULTFONTFILE=\"standard.flf\" figlet.c zipio.c crc.c inflate.c -o figlet chmod a+x figlet gcc -O2 -o chkfont chkfont.c => Unwrapping files-to-be-installed. # # make install ===> Checking for vulnerabilities in figlet-2.2.1nb2 ===> Installing for figlet-2.2.1nb2 install -d -o root -g wheel -m 755 /usr/pkg/bin install -d -o root -g wheel -m 755 /usr/pkg/man/man6 mkdir -p /usr/pkg/share/figlet cp figlet /usr/pkg/bin cp chkfont /usr/pkg/bin chmod 555 figlist showfigfonts cp figlist /usr/pkg/bin cp showfigfonts /usr/pkg/bin cp fonts/*.flf /usr/pkg/share/figlet cp fonts/*.flc /usr/pkg/share/figlet cp figlet.6 /usr/pkg/man/man6 ===> Registering installation for figlet-2.2.1nb2 #Packaging figlet# make package ===> Checking for vulnerabilities in figlet-2.2.1nb2 ===> Packaging figlet-2.2.1nb2 ===> Building binary package for figlet-2.2.1nb2 Creating package /home/cvs/pkgsrc/packages/i386/All/figlet-2.2.1nb2.tgz Using SrcDir value of /usr/pkg Registering depends:. #Layout of the FTP server's package archiveLayout for precompiled binary packages on ftp.NetBSD.org: /pub/NetBSD/packages/ distfiles/ # Unpacked pkgsrc trees pkgsrc-current -> /pub/NetBSD/NetBSD-current/pkgsrc pkgsrc-2003Q4 -> N/A pkgsrc-2004Q1/pkgsrc # pkgsrc archives pkgsrc-current.tar.gz -> ../NetBSD-current/tar_files/pkgsrc.tar.gz pkgsrc-2003Q4.tar.gz -> N/A pkgsrc-2004Q1.tar.gz -> N/A # Per pkgsrc-release/OS-release/arch package archives pkgsrc-2003Q4/ NetBSD-1.6.2/ i386/ All/ archivers/ foo -> ../All/foo ... pkgsrc-2004Q1/ NetBSD-1.6.2/ i386/ All/ ... NetBSD-2.0/ i386/ All/ ... SunOS-5.9/ sparc/ All/ ... x86/ All/ ... # Per os-release package archive convenience links NetBSD-1.6.2 -> 1.6.2 1.6.2/ i386 -> ../pkgsrc-2004Q1/NetBSD-1.6.2/i386 m68k/ All/ archivers/ foo -> ../All/foo ... amiga -> m68k atari -> m68k ... 2.0 -> NetBSD-2.0 # backward compat, historic NetBSD-2.0/ i386 -> ../pkgsrc-2004Q1/NetBSD-2.0/i386 SunOS-5.9/ sparc -> ../pkgsrc-2004Q1/SunOS-5.9/sparc x86 -> ../pkgsrc-2004Q1/SunOS-5.9/x86 To create: Run bulk build, see Upload /usr/pkgsrc/packages to ftp://ftp.NetBSD.org/pub/NetBSD/packages/\ pkgsrc-2004Q3/\ # pkgsrc-branch `uname -s`-`uname -r`/ # OS & version `uname -p` # architecture

If necessary, create a symlink ln -s ‘uname -m‘ ‘uname -p‘ (amiga -> m68k, ...) Editing guidelines for the pkgsrc guide This section contains information on editing the pkgsrc guide itself. Targets The pkgsrc guide’s source code is stored in pkgsrc/doc/guide/files, and several files are created from it: pkgsrc/doc/pkgsrc.txt, which replaces pkgsrc/Packages.txt pkgsrc/doc/pkgsrc.html http://www.NetBSD.org/Documentation/pkgsrc/: the documentation on the NetBSD website will be built from pkgsrc and kept up to date on the web server itself. This means you must make sure that your changes haven’t broken the build! http://www.NetBSD.org/Documentation/pkgsrc/pkgsrc.pdf: PDF version of the pkgsrc guide. http://www.NetBSD.org/Documentation/pkgsrc/pkgsrc.ps: PostScript version of the pkgsrc guide. Procedure The procedure to edit the pkgsrc guide is: Make sure you have the packages needed to re-generate the pkgsrc guide (and other XML-based NetBSD documentation) installed. These are netbsd-doc for creating the ASCII- and HTML-version, and netbsd-doc-printfor the PostScript- and PDF version. You will need both packages installed, to make sure documentation is consistent across all formats. The packages can be found in pkgsrc/meta-pkgs/netbsd-doc and pkgsrc/meta-pkgs/netbsd-doc-print. Edit the XML file(s) in pkgsrc/doc/guide/files. Run make extract && make do-lint in pkgsrc/doc/guide to check the XML syntax, and fix it if needed. Run make in pkgsrc/doc/guide to build the HTML and ASCII version. If all is well, run make install-doc to put the generated files into pkgsrc/doc. cvs commit pkgsrc/doc/guide/files cvs commit -m re-generate pkgsrc/doc/pkgsrc.{html,txt} Until the webserver on www.NetBSD.org is really updated automatically to pick up changes to the pkgsrc guide automatically, also run make install-htdoc HTDOCSDIR=../../../htdocs (or similar, adjust HTDOCSDIR!). cvs commit htdocs/Documentation/pkgsrc