pbuilder User's Manual

Usage and operations

COLLABORATORS			
	TITLE : pbuilder User's Manual		
ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY	Junichi Uekawa	2007-5-27	

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME

Contents

List of Tables

Chapter 1

Introducing pbuilder

1.1 Aims of pbuilder

pbuilder stands for Personal Builder, and it is an automatic Debian Package Building system for personal development workstation environments. **pbuilder** aims to be an easy-to-set-up system for auto-building Debian packages inside a clean-room environment, so that it is possible to verify that a package can be built on most Debian installations. The clean-room environment is achieved through the use of a base chroot image, so that only minimal packages will be installed inside the chroot.

The Debian distribution consists of free software accompanied with source. The source code within Debian's "main" section must build within Debian "main", with only the explicitly specified build-dependencies installed.

The primary aim of **pbuilder** is different from other auto-building systems in Debian in that its aim is not to try to build as many packages as possible. It does not try to guess what a package needs, and in most cases it tries the worst choice of all if there is a choice to be made.

In this way, **pbuilder** tries to ensure that packages tested against **pbuilder** will build properly in most Debian installations, hopefully resulting in a good overall Debian source-buildability.

The goal of making Debian buildable from source is somewhat accomplished, and has seen good progress. In the past age of Debian 3.0, there were many problems when building from source. More recent versions of Debian are much better.

Chapter 2

Using pbuilder

There are several simple commands for operation. **pbuilder create**, **pbuilder update**, and **pbuilder build** commands are the typical commands used. Let us look at the commands one by one.

2.1 Creating a base chroot image tar-ball

pbuilder create will create a base chroot image tar-ball (base.tgz). All other commands will operate on the resulting base.tgz. If the Debian release to be created within chroot is not going to be "sid" (which is the default), the distribution code-name needs to be specified with the **--distribution** command-line option.

debootstrap [1] is used to create the bare minimum Debian installation, and then build-essential packages are installed on top of the minimum installation using the **apt-get** inside the chroot.

For more thorough documentation of command-line options, see the pbuilder(8) manual page. Some configuration will be required for `/etc/pbuilderrc` for the mirror site [2] to use, and proxy configuration may be required to allow access through HTTP. See the pbuilderrc(5) manual page for details.

2.2 Updating the base.tgz

pbuilder update will update the base.tgz. It will extract the chroot, invoke **apt-get update** and **apt-get dist-upgrade** inside the chroot, and then recreate the base.tgz (the base tar-ball).

It is possible to switch the distribution for which the base.tgz is targeted at at this point. Specify **--distribution *sid* --override-config** to change the distribution to sid. [3]

For more thorough documentation of command-line options, see the pbuilder(8) manual page.

2.3 Building a package using the base.tgz

To build a package inside the chroot, invoke **pbuilder build `whatever.dsc`**. **pbuilder** will extract the base.tgz to a temporary working directory, enter the directory with chroot, satisfy the build-dependencies inside chroot, and build the package. The built packages will be moved to a directory specified with the **--buildresult** command-line option.

The **--basetgz** option can be used to specify which base.tgz to use.

[1] debootstrap or cdebootstrap can be chosen

[2] The mirror site should preferably be a local mirror or a cache server, so as not to overload the public mirrors with a lot of access. Use of tools such as apt-proxy would be advisable.

[3] Only upgrading is supported. Debian does not generally support downgrading (yet?).

pbuilder will extract a fresh base chroot image from base.tgz. (base.tgz is created with the **pbuilder create**, and updated with the **pbuilder update**). The chroot is populated with build-dependencies by parsing debian/control and invoking **apt-get**.

For more thorough documentation of command-line options, see the pbuilder(8) manual page

2.4 Facilitating Debian Developers' typing: pdebuild

pdebuild is a little wrapper script that does the most frequent of all tasks. A Debian Developer may try to run **debuild**, and build a package inside a Debian source directory. **pdebuild** will allow similar control, and allow the package to be built inside the chroot, to check that the current source tree will build happily inside the chroot.

pdebuild calls **dpkg-source** to build the source packages, and then invokes **pbuilder** on the resulting source package. However, unlike debuild, the resulting deb files will be found in the **`--buildresult`** directory.

See the pdebuild(1) manual page for more details.

There is a slightly different mode of operation available in **pdebuild** since version 0.97. **pdebuild** usually runs **debian/rules clean** outside of the chroot; however, it is possible to change the behavior to run it inside the chroot with the **`--use-pdebuild-internal`**. It will try to bind mount the working directory inside chroot, and run **dpkg-buildpackage** inside. It has the following characteristics, and is not yet the default mode of operation.

- Satisfies build-dependency inside the chroot before creating source package (which is a good point that default **pdebuild** could not do).
- The working directory is modified from inside the chroot.
- Building with **pdebuild** does not guarantee that it works with **pbuilder**.
- If making the source package fails, the session using the chroot is wasted (chroot creation takes a bit of time, which should be improved with cowdancer).
- Does not work in the same manner as it used to; for example, **`--buildresult`** does not have any effect.
- The build inside chroot is run with the current user outside chroot.

2.5 Configuration Files

It is possible to specify all settings by command-line options. However, for typing convenience, it is possible to use a configuration file.

`/etc/pbuilderrc` and `${HOME}/.pbuilderrc` are read in when **pbuilder** is invoked. The possible options are documented in the pbuilderrc(5) manual page.

It is useful to use the `--configfile` option to load up a preset configuration file when switching between configuration files for different distributions.

Please note that `${HOME}/.pbuilderrc` supersedes system settings. A caveat is that if you have some configuration, you may need to tweak the configuration to work with new versions of pbuilder when upgrading.

2.6 Building packages as non-root inside the chroot

pbuilder requires full root privilege when it is satisfying the build-dependencies, but most packages do not need root privilege to build, or even refuse to build when they are built as root. **pbuilder** can create a user which is only used inside **pbuilder** and use that user id when building, and use the **fakeroot** command when root privilege is required.

The BUILDUSERID configuration option should be set to a value for a user id that does not already exist on the system, so that it is more difficult for packages that are being built with **pbuilder** to affect the environment outside the chroot. When

the BUILDUSERNAME configuration option is also set, **pbuilder** will use the specified user name and fakeroot for building packages, instead of running as root inside chroot.

Even when using the fakerooting method, **pbuilder** will run with root privilege when it is required. For example, when installing packages to the chroot, **pbuilder** will run under root privilege.

To be able to invoke **pbuilder** without being root, you need to use user-mode-linux, as explained in Chapter 3.

2.7 Using pbuilder for back-porting

pbuilder can be used for back-porting software from the latest Debian distribution to the older stable distribution, by using a chroot that contains an image of the older distribution, and building packages inside the chroot. There are several points to consider, and due to the following reasons, automatic back-porting is usually not possible, and manual interaction is required:

- The package from the unstable distribution may depend on packages or versions of packages which are only available in unstable. Thus, it may not be possible to satisfy Build-Depends: on stable (without additional backporting work).
- The stable distribution may have bugs that have been fixed in unstable which need to be worked around.
- The package in the unstable distribution may have problems building even on unstable.

2.8 Mass-building packages

pbuilder can be automated, because its operations are non-interactive. It is possible to run **pbuilder** through multiple packages non-interactively. Several such scripts are known to exist. Junichi Uekawa has been running such a script since 2001, and has been filing bugs on packages that fail the test of **pbuilder**. There were several problems with auto-building:

- Build-Dependencies need to install non-interactively, but some packages are so broken that they cannot install without interaction (like postgresql).
- When a library package breaks, or gcc/gcj/g++ breaks, or even bison, a large number of build failures are reported. (gcj-3.0 had no "javac", bison got more strict, etc.)
- Some people were quite hostile against build failure reports.

Most of the initial bugs have been resolved in the **pbuilder** sweep done around 2002, but these transitional problems which affect a large portion of Debian Archive do arise from time to time. Regression tests have their values.

A script that was used by Junichi Uekawa in the initial run is now included in the **pbuilder** distribution, as **pbuildd.sh**. It is available in `/usr/share/doc/pbuilder/examples/pbuildd/` and its configuration is in `/etc/pbuilder/pbuildd-config.sh`. It should be easy enough to set up for people who are used to **pbuilder**. It has been running for quite a while, and it should be possible to set the application up on your system also. This version of the code is not the most tested, but should function as a starter.

To set up pbuildd, there are some points to be aware of.

- A file `./avoidlist` needs to be available with the list of packages to avoid building.
- It will try building anything, even packages that are not intended for your architecture.
- Because you are running random build scripts, it is better to use the fakeroot option of **pbuilder**, to avoid running the build under root privilege.
- Because not all builds are guaranteed to finish in a finite time, setting a timeout is probably necessary, or pbuildd may stall with a bad build.
- Some packages require a lot of disk space; around 2GB seems to be sufficient for the largest packages for the time being. If you find otherwise, please inform the maintainer of this documentation.

2.9 Auto-backporting scripts

There are some people who use **pbuilder** to automatically back-port a subset of packages to the stable distribution.

I would like some information on how people are doing it. I would appreciate any feedback or information on how you are doing it, or any examples.

2.10 Using pbuilder for automated testing of packages

pbuilder can be used for automated testing of packages. It has the feature of allowing hooks to be placed, and these hooks can try to install packages inside the chroot, or run them, or whatever else that can be done. Some known tests and ideas:

- Automatic install-remove-install-purge-upgrade-remove-upgrade-purge test-suite (distributed as an example, `B91dpkg-i`), or just check that everything installs somewhat (`execute_installtest.sh`).
- Automatically running lintian (distributed as an example in `/usr/share/doc/pbuilder/examples/B90lintian`).
- Automatic debian-test of the package? The debian-test package has been removed from Debian. A **pbuilder** implementation can be found as debian/pbuilder-test directory, implemented through the B92test-pkg script.

To use the B92test-pkg script, first, add it to your hook directory. [4]. The test files are shell scripts placed in `debian/pbuilder-test/NN_name` (where NN is a number) following the run-parts standard[5] for file names. After a successful build, packages are first tested for installation and removal, and then each test is run inside the chroot. The current directory is the top directory of the source-code. This means you can expect to be able to use the ./debian/ directory from inside your scripts.

Example scripts for use with pbuilder-test can be found in `/usr/share/doc/pbuilder/examples/pbuilder-test`.

2.11 Using pbuilder for testing builds with alternate compilers

Most packages are compiled with **gcc** or **g++** and use the default compiler version, which was gcc 2.95 for Debian GNU/Linux 3.0 (i386). However, Debian 3.0 was distributed with other compilers, under package names such as **gcc-3.2** for gcc compiler version 3.2. It was therefore possible to try compiling packages against different compiler versions. **pentium-builder** provides an infrastructure for using a different compiler for building packages than the default gcc, by providing a wrapper script called gcc that calls the real gcc. To use **pentium-builder** in **pbuilder**, it is possible to set up the following in the configuration:

```
EXTRAPACKAGES="pentium-builder gcc-3.2 g++-3.2"
export DEBIAN_BUILDARCH=athlon
export DEBIAN_BUILDGCCVER=3.2
```

It will instruct **pbuilder** to install the **pentium-builder** package and also the GCC 3.2 compiler packages inside the chroot, and set the environment variables required for **pentium-builder** to function.

[4] It is possible to specify a **--hookdir /usr/share/doc/pbuilder/examples** command-line option to include all example hooks as well.
[5] See run-parts(8). For example, no '.' in file names!

Chapter 3

Using User-mode-linux with pbuilder

It is possible to use user-mode-linux by invoking **pbuilder-user-mode-linux** instead of **pbuilder**. **pbuilder-user-mode-linux** doesn't require root privileges, and it uses the copy-on-write (COW) disk access method of **User-mode-linux**, which typically makes it much faster than the traditional **pbuilder**.

User-mode-linux is a somewhat less proven platform than the standard Unix tools that **pbuilder** relies on (**chroot**, **tar**, and **gzip**) but mature enough to support **pbuilder-user-mode-linux** since its version 0.59. And since then, **pbuilder-user-mode-linux** has seen a rapid evolution.

The configuration of **pbuilder-user-mode-linux** goes in three steps:

- Configuration of user-mode-linux
- Configuration of rootstrap
- Configuration of pbuilder-uml

3.1 Configuring user-mode-linux

user-mode-linux isn't completely trivial to set up. It would probably be useful to acquaint yourself with it a bit before attempting to use **rootstrap** or **pbuilder-user-mode-linux**. For details, read `/usr/share/doc/uml-utilities/README.Debian` and the **user-mode-linux** documentation. (It's in a separate package, user-mode-linux-doc.)

user-mode-linux requires the user to be in the uml-net group in order to configure the network unless you are using slirp.

If you compile your own kernel, you may want to verify that you enable TUN/TAP support, and you might want to consider the SKAS patch.

3.2 Configuring rootstrap

rootstrap is a wrapper around debootstrap. It creates a Debian disk image for use with UML. To configure rootstrap, there are several requirements.

- Install the rootstrap package.
- TUN/TAP only: add the user to the uml-net group to allow access to the network

```
adduser dancer uml-net
```

- TUN/TAP only: Check that the kernel supports the TUN/TAP interface, or recompile the kernel if necessary.

- Set up `/etc/rootstrap/rootstrap.conf`. For example, if the current host is 192.168.1.2, changing the following entries to something like this seems to work.

```
transport=tuntap
interface=eth0
gateway=192.168.1.1
mirror=http://192.168.1.2:8081/debian
host=192.168.1.198
uml=192.168.1.199
netmask=255.255.255.0
```

  Some experimentation with configuration and running **rootstrap ~/test.uml** to actually test it would be handy.

  Using slirp requires less configuration. The default configuration comes with a working example.

3.3 Configuring pbuilder-uml

The following needs to happen:

- Install the pbuilder-uml package.
- Set up the configuration file `/etc/pbuilder/pbuilder-uml.conf` in the following manner. It will be different for slirp.

```
MY_ETH0=tuntap,,,192.168.1.198
UML_IP=192.168.1.199
UML_NETMASK=255.255.255.0
UML_NETWORK=192.168.1.0
UML_BROADCAST=255.255.255.255
UML_GATEWAY=192.168.1.1
PBUILDER_UML_IMAGE="/home/dancer/uml-image"
```

  Also, it needs to match the rootstrap configuration.

- Make sure BUILDPLACE is writable by the user. Change BUILDPLACE in the configuration file to a place where the user has access.
- Run **pbuilder-user-mode-linux `create --distribution sid`** to create the image.
- Try running **pbuilder-user-mode-linux build**.

3.4 Considerations for running pbuilder-user-mode-linux

pbuilder-user-mode-linux emulates most of **pbuilder**, but there are some differences.

- **pbuilder-user-mode-linux** does not support all options of **pbuilder** properly yet. This is a problem, and will be addressed as specific areas are discovered.
- /tmp is handled differently inside **pbuilder-user-mode-linux**. In **pbuilder-user-mode-linux**, `/tmp` is mounted as tmpfs inside UML, so accessing files under `/tmp` from outside user-mode-linux does not work. It affects options like **`--configfile`**, and when trying to build packages placed under `/tmp`.

3.5 Parallel running of pbuilder-user-mode-linux

To run **pbuilder-user-mode-linux** in parallel on a system, there are a few things to bear in mind.

- The create and update methods must not be run when a build is in progress, or the COW file will be invalidated.
- If you are not using slirp, user-mode-linux processes that are running in parallel need to have different IP addresses. Just trying to run the **pbuilder-user-mode-linux** several times will result in failure to access the network. But something like the following will work:

```
for IP in 102 103 104 105; do
  xterm -e pbuilder-user-mode-linux build --uml-ip 192.168.0.$IP \
    20030107/whizzytex_1.1.1-1.dsc &
done
```

  When using slirp, this problem does not exist.

3.6 Using pbuilder-user-mode-linux as a wrapper script to start up a virtual machine

It is possible to use **pbuilder-user-mode-linux** for other uses than just building Debian packages. **pbuilder-user-mode-linux `login`** will let a user use a shell inside the user-mode-linux **pbuilder** base image, and **pbuilder-user-mode-linux `execute`** will allow the user to execute a script inside the image.

You can use the script to install ssh and add a new user, so that it is possible to access inside the user-mode-linux through ssh.

Note that it is not possible to use a script from `/tmp` due to the way **pbuilder-user-mode-linux** mounts a tmpfs at `/tmp`.

The following example script may be useful in starting a sshd inside user-mode-linux.

```
#!/bin/bash

apt-get install -y ssh xbase-clients xterm
echo "enter root password"
passwd
cp /etc/ssh/sshd_config{,-}
sed 's/X11Forwarding.*/X11Forwarding yes/' /etc/ssh/sshd_config- > /etc/ssh/sshd_config

/etc/init.d/ssh restart
ifconfig
echo "Hit enter to finish"
read
```

Chapter 4

Frequently asked questions

Here, known problems and frequently asked questions are documented. This portion was initially available in README.Debian file, but moved here.

4.1 pbuilder create fails

It often happens that **pbuilder** cannot create the latest chroot. Try upgrading **pbuilder** and debootstrap. It is currently only possible to create software that handles the past. Future prediction is a feature which may be added later after we have become comfortable with the past.

There are people who occasionally back port debootstrap to stable versions; hunt for them.

When there are errors with the debootstrap phase, the debootstrap script needs to be fixed. **pbuilder** does not provide a way to work around debootstrap.

4.2 Directories that cannot be bind-mounted

Because of the way **pbuilder** works, there are several directories that cannot be bind-mounted when running **pbuilder**. The directories include `/tmp`, `/var/cache/pbuilder`, and system directories such as `/etc` and `/usr`. The recommendation is to use directories under the user's home directory for bind-mounts.

4.3 Logging in to pbuilder to investigate build failure

It is possible to invoke a shell session after a build failure. Example hook scripts are provided as `C10shell` and `C11screen` scripts. The C10shell script will start bash inside chroot, and the C11screen script will start a GNU screen inside the chroot.

4.4 Logging in to pbuilder to modify the environment

It is sometimes necessary to modify the chroot environment. **login** will remove the contents of the chroot after logout. It is possible to invoke a shell using hook scripts. **pbuilder update** executes 'E' scripts, and a sample for invoking a shell is provided as `C10shell`.

```
$ mkdir ~/loginhooks
$ cp C10shell ~/loginhooks/E10shell
$ sudo pbuilder update --hookdir ~/loginhooks/E10shell
```

It is also possible to add `--save-after-exec` and/or `--save-after-login` options to the **pbuilder login** session to accomplish the goal. It is possible to add the `--uml-login-nocow` option to **pbuilder-user-mode-linux `login`** session as well.

4.5 Setting BUILDRESULTUID for sudo sessions

It is possible to set

```
BUILDRESULTUID=$SUDO_UID
```

in pbuilderrc to set the proper BUILDRESULTUID when using **sudo**.

4.6 Notes on usage of $TMPDIR

If you are setting $TMPDIR to an unusual value, of other than `/tmp`, you will find that some errors may occur inside the chroot, such as **dpkg-source** failing.

There are two options: you may install a hook to create that directory, or set

```
export TMPDIR=/tmp
```

in pbuilderrc. Take your pick.

An example script is provided as `examples/D10tmp` with **pbuilder**.

4.7 Creating a shortcut for running pbuilder with a specific distribution

When working with multiple chroots, it would be nice to work with scripts that reduce the amount of typing. An example script, `pbuilder-distribution.sh`, is provided as an example. Invoking the script as `pbuilder-squeeze` will invoke **pbuilder** with a squeeze chroot.

4.8 Using environmental variables for running pbuilder for a specific distribution

This section[1] describes briefly a way to set up and use multiple pbuilder setups by creating a pbuilderrc configuration in your home path (`$HOME/.pbuilderrc`) and using the variable "DIST" when running pbuilder or pdebuild.

First, set up `$HOME/.pbuilderrc` to look like:

```
if [ -n "${DIST}" ]; then
        BASETGZ="`dirname $BASETGZ`/$DIST-base.tgz"
        DISTRIBUTION="$DIST"
        BUILDRESULT="/var/cache/pbuilder/$DIST/result/"
        APTCACHE="/var/cache/pbuilder/$DIST/aptcache/"
fi
```

Then, whenever you wish to use pbuilder for a particular distro, assign a value to "DIST" that is one of the distros available for Debian or any Debian based distro you happen to be running (i.e., whatever is found under /usr/lib/debootstrap/scripts).

Here are some examples for running pbuilder or pdebuild:

```
DIST=gutsy sudo pbuilder create

DIST=sid sudo pbuilder create --mirror http://http.us.debian.org/debian

DIST=gutsy sudo pbuilder create \
        --othermirror "deb http://archive.ubuntu.com/ubuntu gutsy universe \
```

[1] This part of the documentation contributed by Andres Mejia.
This example was taken from a wiki (https://wiki.ubuntu.com/PbuilderHowto).

```
        multiverse"

DIST=gutsy sudo pbuilder update

DIST=sid sudo pbuilder update --override-config --mirror \
http://http.us.debian.org/debian \
--othermirror "deb http://http.us.debian.org/debian sid contrib non-free"

DIST=gutsy pdebuild
```

4.9 Using special apt sources lists, and local packages

If you have some very specialized requirements on your apt setup inside **pbuilder**, it is possible to specify them through the **--othermirror** option. Try something like: **--othermirror "deb http://local/mirror stable main|deb-src http://local/source/repository ./"**

To use the local file system instead of HTTP, it is necessary to do bind-mounting. **--bindmounts** is a command-line option useful for such cases.

It might be convenient to use your built packages from inside the chroot. It is possible to automate the task with the following configuration. First, set up pbuilderrc to bindmount your build results directory.

```
BINDMOUNTS="/var/cache/pbuilder/result"
```

Then, add the following hook

```
# cat /var/cache/pbuilder/hooks/D70results
#!/bin/sh
cd /var/cache/pbuilder/result/
/usr/bin/dpkg-scanpackages . /dev/null > /var/cache/pbuilder/result/Packages
/usr/bin/apt-get update
```

This way, you can use `deb file:/var/cache/pbuilder/result`

To add a new apt-key inside chroot:

```
sudo pbuilder --login --save-after-login
# apt-key add - <<EOF
...public key goes here...
EOF
# logout
```

4.10 How to get pbuilder to run apt-get update before trying to satisfy build-dependencie

You can use hook scripts for this. D scripts are run before satisfying build-dependency.

This snippet comes from Ondrej Sury.

4.11 Different bash prompts inside pbuilder login

To make distinguishing bash prompts inside **pbuilder** easier, it is possible to set environment variables such as PS1 inside `pbuilderrc`

With versions of bash more recent than 2.05b-2-15, the value of the debian_chroot variable, if set, is included in the value of PS1 (the Bash prompt) inside the chroot. In prior versions of bash,[2] setting PS1 in pbuilderrc worked.

Example of debian_chroot:

[2] Versions of bash from and before Debian 3.0.

```
export debian_chroot="pbuild$$"
```

Example of PS1:

```
export PS1="pbuild chroot 32165 # "
```

4.12 Creating a chroot reminder

Bash prompts will help you remember that you are inside a chroot. There are other cases where you may want other signs of being inside a chroot. Check out the `examples/F90chrootmemo` hook script. It will create a file called `/CHROOT` inside your chroot.

4.13 Using /var/cache/apt/archives for the package cache

For the help of low-bandwidth systems, it is possible to use `/var/cache/apt/archives` as the package cache. Just specify it instead of the default `/var/cache/pbuilder/aptcache`.

It is however not possible to do so currently with the user-mode-linux version of **pbuilder**, because `/var/cache/apt/archives` is usually only writable by root.

Use of dedicated tools such as apt-proxy is recommended, since caching of packages would benefit the system outside the scope of **pbuilder**.

4.14 pbuilder back ported to stable Debian releases

Currently, a stable back port of pbuilder is available at backports.org.

4.15 Warning about LOGNAME not being defined

You might see a lot of warning messages when running **pbuilder**.

```
dpkg-genchanges: warning: no utmp entry available and LOGNAME not defined; using uid of ↩
    process (1234)
```

It is currently safe to ignore this warning message. Please report back if you find any problem with having LOGNAME unset. Setting LOGNAME caused a few problems when invoking **chroot**. For example, dpkg requires getpwnam to succeed inside chroot, which means LOGNAME and the related user information have to be set up inside chroot.

4.16 Cannot Build-conflict against an essential package

pbuilder does not currently allow Build-Conflicts against essential packages. It should be obvious that essential packages should not be removed from a working Debian system, and a source package should not try to force removal of such packages on people building the package.

4.17 Avoiding the "ln: Invalid cross-device link" message

By default, **pbuilder** uses hard links to manage the **pbuilder** package cache. It is not possible to make hard links across different devices; and thus this error will occur, depending on your set up. If this happens, set

```
APTCACHEHARDLINK=no
```

in your pbuilderrc file. Note that packages in **APTCACHE** will be copied into the chroot local cache, so plan for enough space on the **BUILDPLACE** device.

4.18 Using fakechroot

It is possible to use **fakechroot** instead of being root to run **pbuilder**; however, several things make this impractical. **fakechroot** overrides library loads and tries to override default libc functions when providing the functionality of virtual **chroot**. However, some binaries do not use libc to function, or override the overriding provided by **fakechroot**. One example is **ldd**. Inside **fakechroot**, **ldd** will check the library dependency outside of the chroot, which is not the expected behavior.

To work around the problem, debootstrap has a `--variant fakechroot` option. Use that, so that ldd and ldconfig are overridden.

Make sure you have set your LD_PRELOAD path correctly, as described in the fakechroot manpage.

4.19 Using debconf inside pbuilder sessions

To use debconf inside **pbuilder**, setting DEBIAN_FRONTEND to "readline" in `pbuilderrc` should work. Setting it to "dialog" should also work, but make sure whiptail or dialog is installed inside the chroot.

4.20 nodev mount options hinder pbuilder activity

If you see messages such as this when building a chroot, you are mounting the file system with the nodev option.

```
  /var/lib/dpkg/info/base-files.postinst: /dev/null: Permission denied
```

You will also have problems if you mount the file system with the noexec option, or nosuid. Make sure you do not have these flags set when mounting the file system for `/var/cache/pbuilder` or $BUILDPLACE.

This is not a problem when using **user-mode-linux**.

See 316135 for example.

4.21 pbuilder is slow

pbuilder is often slow. The slowest part of **pbuilder** is extracting the tar.gz every time **pbuilder** is invoked. That can be avoided by using **pbuilder-user-mode-linux**. **pbuilder-user-mode-linux** uses the COW file system, and thus does not need to clean up and recreate the root file system.

pbuilder-user-mode-linux is slower in executing the actual build system, due to the usual **user-mode-linux** overhead for system calls. It is more friendly to the hard drive.

pbuilder with cowdancer is also an alternative that improves the speed of pbuilder startup.

4.22 Using pdebuild to sponsor package

To sign a package marking it for sponsorship, it is possible to use **--auto-debsign** and **--debsign-k** options of **pdebuild**.

```
  pdebuild  --auto-debsign  --debsign-k XXXXXXXX
```

4.23 Why is there a source.changes file in ../?

When running **pdebuild**, **pbuilder** will run dpkg-buildpackage to create a Debian source package to pass it on to **pbuilder**. A file named XXXX_YYY_source.changes is what remains from that process. It is harmless unless you try to upload it to the Debian archive.

This behavior is different when running through `--use-pdebuild-internal`

4.24 amd64 and i386-mode

amd64 architectures are capable of running binaries in i386 mode. It is possible to use **pbuilder** to run packages, using **linux32** and the **debootstrap --arch** option. Specifically, a command-line option like the following will work.

```
pbuilder create --distribution sid --debootstrapopts --arch --debootstrapopts i386 \
  --basetgz /var/cache/pbuilder/base-i386.tgz --mirror http://ftp.jp.debian.org/debian
linux32 pbuilder build --basetgz /var/cache/pbuilder/base-i386.tgz
```

4.25 Using tmpfs for buildplace

To improve speed of operation, it is possible to use tmpfs for the pbuilder build location. Mount tmpfs to `/var/cache/pbuilder/build`, and set

```
APTCACHEHARDLINK=no
```

.

4.26 Using svn-buildpackage together with pbuilder

The pdebuild command can be used with the svn-buildpackage --svn-builder command-line option: [3]

```
alias svn-cowbuilder="svn-buildpackage --svn-builder='pdebuild --pbuilder cowbuilder"
```

[3] Zack has posted an example on his blog.

Chapter 5

Troubleshooting and development

5.1 Reporting bugs

To report bugs, it would be important to have a log of what's going wrong. Most of the time, adding a **`--debug`** option and re-running the session should do the trick. Please send the log of such a session along with your problem to ease the debugging process.

5.2 Mailing list

There is a mailing list for **pbuilder** on alioth (pbuilder-maint@lists.alioth.debian.org). You can subscribe through the alioth web interface: http://alioth.debian.org/mail/?group_id=30778.

5.3 IRC Channel

For coordination and communication, IRC channel #pbuilder on irc.oftc.net is used. Please log your intent there when you are going to start doing some changes and committing some change.

5.4 Information for pbuilder developers

This section tries to document current development practices and how things generally operate in development.

pbuilder is co-maintained with resources provided by Alioth. There is an Alioth project page at http://alioth.debian.org/projects/pbuilder. A home page is also available, at http://alioth.debian.org/projects/pbuilder, which shows this text. A git repository is available through http, git, or (if you have an account on alioth,) ssh.

```
git-clone git://git.debian.org/git/pbuilder/pbuilder.git
git-clone http://git.debian.org/git/pbuilder/pbuilder.git
git-clone ssh://git.debian.org/git/pbuilder/pbuilder.git
```

Git commit message should have the first line describing what the commit does, formatted in the way debian/changelog is formatted because it is copied verbatim to changelog via git-dch. The second line is empty, and the rest should describe the background and extra information related to implementation of the commit.

Test-suites are available in the `./testsuite/` directory. Changes are expected not to break the test-suites. `./run-test.sh` is a basic test-suite, which puts a summary in `run-test.log`, and `run-test-cdebootstrap.log`. `./run-test-regressi sh` is a regression test-suite, which puts the result in `run-test-regression.log`. Currently, run-test.sh is run automatically daily to ensure that pbuilder is working.

Directory	Meaning
./testsuite/	Directory for testsuite.
./testsuite/run-test.sh	Daily regression test to test against Debian Archive changes breaking pbuilder.
./testsuite/run-test.log	A summary of testsuite.
./testsuite/normal/	Directory for testsuite results of running pbuilder with debootstrap.
./testsuite/cdebootstrap/	Directory for testsuite results of running pbuilder with cdebootstrap.
./testsuite/run-regression.sh	Regression testsuite, run every time change is made to pbuilder to make sure there is no regression.
./testsuite/run-regression.log	Summary of test result.
./testsuite/regression/BugID-*.sh	Regression tests, exit 0 for success, exit 1 for failure.
./testsuite/regression/BugID-*	Files used for the regression testsuite.
./testsuite/regression/log/BugID-*.sh.log	Output of the regression test; output from the script is redirected by run-regression.sh.

Table 5.1: Directory structure of the testsuite

When making changes, they should be documented in the Git commit log. git-dch will generate debian/changelog from the commit log. Make the first line of your commit log meaningful, and add any bug-closing information available. debian/changelog should not be edited directly except when releasing a new version.

A TODO file is available in debian/TODO. It's mostly not well-maintained, but hopefully it will be more up-to-date when people start using it. Emacs todoo-mode is used in editing the file.

When releasing a new version of **pbuilder**, the version is tagged with the git tag X.XXX (version number). This is done with the **./git-tag.sh** script, available in the source tree.

Chapter 6

Other uses of pbuilder

6.1 Using pbuilder for small experiments

There are cases when some small amount of experimenting is required, and you do not want to damage the main system, like when installing experimental library packages, or compiling with experimental compilers. For such cases, the **pbuilder login** command is available.

pbuilder login is a debugging feature for **pbuilder** itself, but it also allows users to have a temporary chroot.

Note that the chroot is cleaned after logging out of the shell, and mounting file systems inside it is considered harmful.

6.2 Running little programs inside the chroot

To facilitate using **pbuilder** for other uses, **pbuilder execute** is available. **pbuilder execute** will take a script specified in the command-line argument, and invoke the script inside the chroot.

The script can be useful for sequences of operations such as installing ssh and adding a new user inside the chroot.

Chapter 7

Experimental or wishlist features of pbuilder

There are some advanced features, above that of the basic feature of **pbuilder**, for some specific purposes.

7.1 Using LVM

LVM2 has a useful snapshot function that features Copy-on-write images. That could be used for **pbuilder** just as it can be used for the user-mode-linux **pbuilder** port. The lvmpbuilder script in the examples directory implements such a port. The scripts and documentation can be found under `/usr/share/doc/pbuilder/examples/lvmpbuilder/`.

7.2 Using cowdancer

cowdancer allows copy-on-write semantics on a file system using hard links and hard-link-breaking-on-write tricks. **pbuilder** using **cowdancer** seems to be much faster and it is one ideal point for improvement. **cowbuilder**, a wrapper for **pbuilder** that uses **cowdancer**, is available from the **cowdancer** package since version 0.14.

Example command-lines for cowbuilder look like the following:

```
# cowbuilder --create --distribution sid
# cowbuilder --update --distribution sid
# cowbuilder --build XXX.dsc
```

It is also possible to use cowdancer with the pdebuild command. Specify this with command-line option `--pbuilder` or set it in the PDEBUILD_PBUILDER configuration option.

```
$ pdebuild --pbuilder cowbuilder
```

7.3 Using pbuilder without tar.gz

The **`--no-targz`** option of **pbuilder** will allow usage of **pbuilder** in a different way than conventional usage. It will try to use an existing chroot, and will not try to clean up after working on it. It is an operation mode more like **sbuild**.

It should be possible to create base chroot images for **dchroot** with the following commands:

```
# pbuilder create --distribution lenny --no-targz --basetgz /chroot/lenny
# pbuilder create --distribution squeeze --no-targz --basetgz /chroot/squeeze
# pbuilder create --distribution sid --no-targz --basetgz /chroot/sid
```

7.4 Using pbuilder in a vserver

It is possible to use **pbuilder** in a vserver environment. This requires either vserver-patches in version 2.1.1-rc14 or higher, or a Linux kernel version 2.6.16 or higher.

To use **pbuilder** in a vserver, you need to set the **secure_mount CAPS** in the **ccapabilities** of this vserver.

7.5 Usage of ccache

It is possible to use the C compiler cache **ccache** to speed up repeated builds of the same package (or packages that compile the same files multiple times for some reason). Using **ccache** can speed up repeated building of large packages dramatically, at the cost of some disk space and bookkeeping.

To enable usage of **ccache** with **pbuilder**, you should set CCACHEDIR in your pbuilderrc file.

Current implementation of ccache support has several bugs, such that CCACHEDIR must be owned by the pbuilder build user, and parallel runs of pbuilder are not supported. Therefore it is not enabled by default.

Chapter 8

Reference materials

8.1 Directory structure outside the chroot

Directory	Meaning
/etc/pbuilderrc	Configuration file.
/usr/share/pbuilder/pbuilderrc	Default configuration.
/var/cache/pbuilder/base.tgz	Default location pbuilder uses for base.tgz, the tar-ball containing a basic Debian installation with only the build-essential packages.
/var/cache/pbuilder/build/PID/	Default location pbuilder uses for chroot.
/var/cache/pbuilder/aptcache	Default location **pbuilder** will use as apt cache, to store deb packages required during **pbuilder** build.
/var/cache/pbuilder/ccache	Default location **pbuilder** will use as cache location.
/var/cache/pbuilder/result	Default location where **pbuilder** puts the deb files and other files created after build.
/var/cache/pbuilder/pbuilder-umlresult	Default location where **pbuilder-user-mode-linux** puts the deb files and other files created after build.
/var/cache/pbuilder/pbuilder-mnt	Default location **pbuilder-user-mode-linux** uses for mounting the COW file system, for chrooting.
/tmp	**pbuilder-user-mode-linux** will mount tmpfs for work.
${HOME}/tmp/PID.cow	**pbuilder-user-mode-linux** uses this directory for location of COW file system.
${HOME}/uml-image	**pbuilder-user-mode-linux** uses this directory for user-mode-linux full disk image.

Table 8.1: Directory Structure outside the chroot

8.2 Directory structure inside the chroot

Directory	Meaning
/etc/mtab	Symlink to /proc/mounts.
/tmp/buildd	Default place used in **pbuilder** to place the Debian package to be processed. /tmp/buildd/packagename-version/ will be the root directory of the package being processed. HOME environment variable is set to this value inside chroot by pbuilder-buildpackage. --inputfile will place files here.
/runscript	The script passed as an argument to **pbuilder** execute is passed on.
/tmp/hooks	The location of hooks.
/var/cache/apt/archives	**pbuilder** copies the content of this directory to and from the aptcache directory located outside chroot.
/var/cache/pbuilder/ccache	**pbuilder** bind-mounts this directory for use by ccache.
/tmp/XXXX	**pbuilder-user-mode-linux** uses a script in /tmp to bootstrap into user-mode-linux.

Table 8.2: Directory Structure inside the chroot

Chapter 9

Minor archaeological details

9.1 Documentation history

This document was started on 28 Dec 2002 by Junichi Uekawa, trying to document what is known about **pbuilder**.

This documentation is available from the **pbuilder** source tar-ball, and from the git repository of **pbuilder** (web-based access is possible). A copy of this documentation can be found on the Alioth project page for pbuilder. There is also a PDF version. The homepage for **pbuilder** is http://pbuilder.alioth.debian.org/, hosted by the alioth project.

Documentation is written using DocBook XML, with emacs PSGML mode, and using wysidocbookxml for live previewing.

9.2 Possibly inaccurate Background History of pbuilder

The following is a most possibly inaccurate account of how **pbuilder** came to happen, and other attempts to make something like **pbuilder** happen. This part of the document was originally in the AUTHORS file, to give credit to those who existed before **pbuilder**.

9.2.1 The Time Before pbuilder

There was once dbuild, which was a shell script to build Debian packages from source. Lars Wirzenius wrote that script, and it was good, short, and simple (probably). There was nothing like build-depends then (I think), and it was simple. It might have been improved; I could only find references and no actual source.

debbuild was probably written by James Troup. I don't know because I have never seen the actual code; I could only find some references to it on the net, and mailing list logs.

sbuild is a perl script to build Debian packages from source. It parses Build-Depends, and performs other miscellaneous checks, and has a lot of hacks to actually get things building, including a table of what package to use when virtual packages are specified (does it do that still?). It supports the use of a local database for packages that do not have build-dependencies. It was written by Ronan Hodek, and I think it was patched and fixed and extended by several people. It is part of wanna-build, and used extensively in the Debian buildd system. I think it was maintained mostly by Ryan Murray.

9.2.2 Birth of pbuilder

wanna-build (sbuild) was (at the time of year 2001) quite difficult to set up, and it was never a Debian package. dbuild was something that predated Build-Depends.

Building packages from source using Build-Depends information within a chroot sounded trivial; and **pbuilder** was born. It was initially a shell script with only a few lines, which called debootstrap and chroot and dpkg-buildpackage in the same run, but soon, it was decided that that's too slow.

Yes, and it took almost a year to get things somewhat right, and in the middle of the process, Debian 3.0 was released. Yay. Debian 3.0 wasn't completely buildable with **pbuilder**, but the amount of packages which are not buildable is steadily decreasing (I hope).

9.2.3 And the second year of its life

Someone wanted **pbuilder** to not run as root, and as User-mode-linux has become more useful as time passed, I've started experimenting with **pbuilder-user-mode-linux**. **pbuilder-user-mode-linux** has not stayed functional as much as I would have liked, and bootstrapping the **user-mode-linux** environment has been pretty hard, due to the quality of user-mode-linux code or packaging at that time, which kept on breaking network support in one way or the other.

9.2.4 Fifth year of pbuilder

pbuilder is now widely adopted as an 'almost standard' tool for testing packages, and building packages in a pristine environment. There are other, similar tools that do similar tasks, but they do not share the exact same goal. To commemorate this fact, **pbuilder** is now co-maintained by several people.

sbuild is now a well-maintained Debian package within Debian, and with **pbuilder** being such a slow monster, some people prefer the approach of sbuild. Development to use LVM-snapshots, cowloop, or cowdancer are hoped to improve the situation somewhat.