## NAME

debuild – build a Debian package

## SYNOPSIS

**debuild** [*debuild options*] [*dpkg-buildpackage options*] [**−−lintian-opts** *lintian options*]
**debuild** [*debuild options*] **−−** **binary|binary-arch|binary-indep|clean** ...

## DESCRIPTION

**debuild** creates all the files necessary for uploading a Debian package. It first runs **dpkg-buildpackage**, then runs **lintian** on the *.changes* file created (assuming that **lintian** is installed), and finally signs the appropriate files (using **debsign**(1) to do this instead of **dpkg-buildpackage**(1) itself; all relevant key-signing options are passed on). Signing will be skipped if the distribution is *UNRELEASED*, unless **dpkg-buildpackage**'s **−−force-sign** option is used. Parameters can be passed to **dpkg-buildpackage** and **lintian**, where the parameters to the latter are indicated with the **−−lintian-opts** option. The allowable options in this case are **−−lintian** and **−−no-lintian** to force or skip the **lintian** step, respectively. The default is to run **lintian**. There are also various options available for setting and preserving environment variables, as described below in the Environment Variables section. In this method of running **debuild**, we also save a build log to the file *../<package>_<version>_<arch>.build*.

An alternative way of using **debuild** is to use one or more of the parameters **binary**, **binary-arch**, **binary-indep** and **clean**, in which case **debuild** will attempt to gain root privileges and then run *debian/rules* with the given parameters. A **−−rootcmd=***gain-root-command* or **−r***gain-root-command* option may be used to specify a method of gaining root privileges. The *gain-root-command* is likely to be one of *fakeroot*, *sudo* or *super*. See below for further discussion of this point. Again, the environment preservation options may be used. In this case, **debuild** will also attempt to run **dpkg-checkbuilddeps** first; this can be explicitly requested or switched off using the options **−D** and **−d** respectively. Note also that if either of these or a **−r** option is specified in the configuration file option **DEBUILD_DPKG_BUILDPACKAGE_OPTS**, then it will be recognised even in this method of invocation of **debuild**.

**debuild** also reads the **devscripts** configuration files as described below. This allows default options to be given.

### Directory name checking

In common with several other scripts in the **devscripts** package, **debuild** will climb the directory tree until it finds a *debian/changelog* file before attempting to build the package. As a safeguard against stray files causing potential problems, it will examine the name of the parent directory once it finds the *debian/changelog* file, and check that the directory name corresponds to the package name. Precisely how it does this is controlled by two configuration file variables **DEVSCRIPTS_CHECK_DIRNAME_LEVEL** and **DEVSCRIPTS_CHECK_DIRNAME_REGEX**, and their corresponding command-line options **−−check-dirname-level** and **−−check-dirname-regex**.

**DEVSCRIPTS_CHECK_DIRNAME_LEVEL** can take the following values:

**0** Never check the directory name.

**1** Only check the directory name if we have had to change directory in our search for *debian/changelog*. This is the default behaviour.

**2** Always check the directory name.

The directory name is checked by testing whether the current directory name (as determined by **pwd**(1)) matches the regex given by the configuration file option **DEVSCRIPTS_CHECK_DIRNAME_REGEX** or by the command line option **−−check-dirname-regex** *regex*. Here *regex* is a Perl regex (see **perlre**(3perl)), which will be anchored at the beginning and the end. If *regex* contains a '/', then it must match the full directory path. If not, then it must match the full directory name. If *regex* contains the string ´PACKAGE', this will be replaced by the source package name, as determined from the *changelog*. The default value for the regex is: ´PACKAGE(-.+)?', thus matching directory names such as PACKAGE and PACKAGE-version.

## ENVIRONMENT VARIABLES

As environment variables can affect the building of a package, often unintentionally, **debuild** sanitises the environment by removing all environment variables except for **TERM**, **HOME**, **LOGNAME**, **GNUPGHOME**, **PGPPATH**, **GPG_AGENT_INFO**, **GPG_TTY**, **DBUS_SESSION_BUS_ADDRESS**, **FAKEROOTKEY**, **DEBEMAIL**, **DEB_***, the (**C**, **CPP**, **CXX**, **LD** and **F)FLAGS** variables and their **_APPEND** counterparts and the locale variables **LANG** and **LC_***. **TERM** is set to 'dumb' if it is unset, and **PATH** is set to "/usr/sbin:/usr/bin:/sbin:/bin:/usr/bin/X11".

If a particular environment variable is required to be passed through untouched to the build process, this may be specified by using a **−−preserve-envvar** *envvar* (which can also be written as **−e** *envvar* option). The environment may be left untouched by using the **−−preserve-env** option. However, even in this case, the **PATH** will be set to the sane value described above. The **only** way to prevent **PATH** from being reset is to specify a **−−preserve-envvar PATH** option. But you are warned that using programs from non-standard locations can easily result in the package being broken, as it will not be able to be built on standard systems.

Note that one may add directories to the beginning of the sanitised **PATH**, using the **−−prepend−path** option. This is useful when one wishes to use tools such as **ccache** or **distcc** for building.

It is also possible to avoid having to type something like *FOO*=*bar* **debuild −e** *FOO* by writing **debuild −e** *FOO*=*bar* or the long form **debuild −−set−envvar** *FOO*=*bar*.

## SUPERUSER REQUIREMENTS

**debuild** needs to be run as superuser to function properly. There are three fundamentally different ways to do this. The first, and preferable, method is to use some root-gaining command. The best one to use is probably **fakeroot**(1), since it does not involve granting any genuine privileges. **super**(1) and **sudo**(1) are also possibilities. If no **−r** (or **−−rootcmd**) option is given (and recall that **dpkg-buildpackage** also accepts a **−r** option) and neither of the following methods is used, then **−rfakeroot** will silently be assumed.

The second method is to use some command such as **su**(1) to become root, and then to do everything as root. Note, though, that **lintian** will abort if it is run as root or setuid root; this can be overcome using the **−−allow-root** option of **lintian** if you know what you are doing.

The third possible method is to have **debuild** installed as setuid root. This is not the default method, and will have to be installed as such by the system administrator. It must also be realised that anyone who can run **debuild** as root or setuid root has **full access to the whole machine**. This method is therefore not recommended, but will work. **debuild** could be installed with mode 4754, so that only members of the owning group could run it. A disadvantage of this method would be that other users would then not be able to use the program. There are many other variants of this option involving multiple copies of **debuild**, or the use of programs such as **sudo** or **super** to grant root privileges to users selectively. If the sysadmin wishes to do this, she should use the **dpkg-statoverride** program to change the permissions of */usr/bin/debuild*. This will ensure that these permissions are preserved across upgrades.

## HOOKS

**debuild** supports a number of hooks when running **dpkg−buildpackage**. Note that the hooks **dpkg-buildpackage** to **lintian** (inclusive) are passed through to **dpkg-buildpackage** using its corresponding **−−hook-***name* option. The available hooks are as follows:

**dpkg-buildpackage-hook**

Run before **dpkg-buildpackage** begins by calling **dpkg-checkbuilddeps**.

Hook is run inside the unpacked source.

Corresponds to **dpkg**'s **init** hook.

**clean-hook**

Run before **dpkg-buildpackage** runs **debian/rules clean** to clean the source tree. (Run even if the tree is not being cleaned because **−nc** is used.)

Hook is run inside the unpacked source.

Corresponds to **dpkg**'s **preclean** hook.

**dpkg-source-hook**
Run after cleaning the tree and before running **dpkg-source**. (Run even if **dpkg-source** is not being called because **−b**, **−B**, or **−A** is used.)

Hook is run inside the unpacked source.

Corresponds to **dpkg**'s **source** hook.

**dpkg-build-hook**
Run after **dpkg-source** and before calling **debian/rules build**. (Run even if this is a source-only build, so **debian/rules build** is not being called.)

Hook is run inside the unpacked source.

Corresponds to **dpkg**'s **build** hook.

**dpkg-binary-hook**
Run between **debian/rules build** and **debian/rules binary**(**−arch**). Run **only** if a binary package is being built.

Hook is run inside the unpacked source.

Corresponds to **dpkg**'s **binary** hook.

**dpkg-genchanges-hook**
Run after the binary package is built and before calling **dpkg-genchanges**.

Hook is run inside the unpacked source.

Corresponds to **dpkg**'s **changes** hook.

**final-clean-hook**
Run after **dpkg-genchanges** and before the final **debian/rules clean**. (Run even if we are not cleaning the tree post-build, which is the default.)

Hook is run inside the unpacked source.

Corresponds to **dpkg**'s **postclean** hook.

**lintian-hook**
Run (once) before calling **lintian**. (Run even if we are not calling **lintian**.)

Hook is run from parent directory of unpacked source.

Corresponds to **dpkg**'s **check** hook.

**signing-hook**
Run after calling **lintian** before any signing takes place. (Run even if we are not signing anything.)

Hook is run from parent directory of unpacked source.

Corresponds to **dpkg**'s **sign** hook, but is run by **debuild**.

**post-dpkg-buildpackage-hook**
Run after everything has finished.

Hook is run from parent directory of unpacked source.

Corresponds to **dpkg**'s **done** hook, but is run by **debuild**.

A hook command can be specified either in the configuration file as, for example, DEBUILD_SIGNING_HOOK='foo' (note the hyphens change into underscores!) or as a command line option **−−signing−hook-foo**. The command will have certain percent substitutions made on it: **%%** will be replaced by a single **%** sign, **%p** will be replaced by the package name, **%v** by the package version number, **%s** by the source version number, **%u** by the upstream version number. Neither **%s** nor **%u** will contain an epoch. **%a** will be **1** if the immediately following action is to be performed and **0** if not (for example, in the **dpkg-**

**source** hook, **%a** will become **1** if **dpkg-source** is to be run and **0** if not). Then it will be handed to the shell to deal with, so it can include redirections and stuff. For example, to only run the **dpkg-source** hook if **dpkg-source** is to be run, the hook could be something like: "if [ %a −eq 1 ]; then ...; fi".

**Please take care with hooks**, as misuse of them can lead to packages which FTBFS (fail to build from source). They can be useful for taking snapshots of things or the like.

## OPTIONS

For details, see above.

**−−no-conf**, **−−noconf**
Do not read any configuration files. This can only be used as the first option given on the command-line.

**−−rootcmd=***gain-root-command*, **−r***gain-root-command*
Command to gain root (or fake root) privileges.

**−−preserve−env**
Do not clean the environment, except for PATH.

**−−preserve−envvar=***var*, **−e***var*
Do not clean the *var* variable from the environment.

If *var* ends in an asterisk ("*") then all variables with names that match the portion of *var* before the asterisk will be preserved.

**−−set−envvar=***var***=***value*, **−e***var***=***value*
Set the environment variable *var* to *value* and do not remove it from the environment.

**−−prepend−path=***value*
Once the normalized PATH has been set, prepend *value* to it.

**−−lintian**
Run **lintian** after **dpkg-buildpackage**. This is the default behaviour, and it overrides any configuration file directive to the contrary.

**−−no−lintian**
Do not run **lintian** after **dpkg-buildpackage**.

**−−no−tgz−check**
Even if we're running **dpkg-buildpackage** and the version number has a Debian revision, do not check that the *.orig.tar.gz* file or *.orig* directory exists before starting the build.

**−−tgz−check**
If we're running **dpkg-buildpackage** and the version number has a Debian revision, check that the *.orig.tar.gz* file or *.orig* directory exists before starting the build. This is the default behaviour.

**−−username** *username*
When signing, use **debrsign** instead of **debsign**. *username* specifies the credentials to be used.

**−−***foo***−hook=***hook*
Set a hook as described above. If *hook* is blank, this unsets the hook.

**−−clear−hooks**
Clears all hooks. They may be reinstated by later command line options.

**−−check-dirname-level** *N*
See the above section **Directory name checking** for an explanation of this option.

**−−check-dirname-regex** *regex*
See the above section **Directory name checking** for an explanation of this option.

**−d** Do not run **dpkg-checkbuilddeps** to check build dependencies.

**−D** Run **dpkg-checkbuilddeps** to check build dependencies.

## CONFIGURATION VARIABLES

The two configuration files */etc/devscripts.conf* and *~/.devscripts* are sourced by a shell in that order to set configuration variables. Command line options can be used to override some of these configuration file settings, otherwise the **−−no−conf** option can be used to prevent reading these files. Environment variable settings are ignored when these configuration files are read. The currently recognised variables are:

**DEBUILD_PRESERVE_ENV**
If this is set to *yes*, then it is the same as the **−−preserve−env** command line parameter being used.

**DEBUILD_PRESERVE_ENVVARS**
Which environment variables to preserve. This should be a comma-separated list of variables. This corresponds to using possibly multiple **−−preserve−envvar** or **−e** options.

**DEBUILD_SET_ENVVAR_***var*=*value*
This corresponds to **−−set−envvar=***var*=*value*.

**DEBUILD_PREPEND_PATH**
This corresponds to **−−prepend−path**.

**DEBUILD_ROOTCMD**
Setting this variable to *prog* is the equivalent of **−r***prog*.

**DEBUILD_TGZ_CHECK**
Setting this variable to *no* is the same as the **−−no−tgz−check** command line option.

**DEBUILD_SIGNING_USERNAME**
Setting this variable is the same as using the **−−username** command line option.

**DEBUILD_DPKG_BUILDPACKAGE_OPTS**
These are options which should be passed to the invocation of **dpkg-buildpackage**. They are given before any command-line options. Due to issues of shell quoting, if a word containing spaces is required as a single option, extra quotes will be required. For example, to ensure that your own GPG key is always used, even for sponsored uploads, the configuration file might contain the line:

DEBUILD_DPKG_BUILDPACKAGE_OPTS="−k'Julian Gilbey <jdg@debian.org>' −sa"

which gives precisely two options. Without the extra single quotes, **dpkg-buildpackage** would reasonably complain that *Gilbey* is an unrecognised option (it doesn't start with a **−** sign).

Also, if this option contains any **−r**, **−d** or **−D** options, these will always be taken account of by **debuild**. Note that a **−r** option in this variable will override the setting in **DEBUILD_ROOTCMD**.

**DEBUILD_*FOO*_HOOK**
The hook variable for the *foo* hook. See the section on hooks above for more details. By default, this is empty.

**DEBUILD_LINTIAN**
Should we run **lintian**? If this is set to *no*, then **lintian** will not be run.

**DEBUILD_LINTIAN_OPTS**
These are options which should be passed to the invocation of **lintian**. They are given before any command-line options, and the usage of this variable is as described for the **DEBUILD_DPKG_BUILDPACKAGE_OPTS** variable.

**DEVSCRIPTS_CHECK_DIRNAME_LEVEL**, **DEVSCRIPTS_CHECK_DIRNAME_REGEX**
See the above section **Directory name checking** for an explanation of these variables. Note that these are package-wide configuration variables, and will therefore affect all **devscripts** scripts which check their value, as described in their respective manpages and in **devscripts.conf**(5).

## EXAMPLES

To build your own package, simply run **debuild** from inside the source tree. **dpkg-buildpackage**(1) options may be given on the command line.

The typical command line options to build only the binary package(s) without signing the .changes file (or the non-existent .dsc file):

debuild −i −us −uc −b

Change the **−b** to **−S** to build only a source package.

An example using **lintian** to check the resulting packages and passing options to it:

debuild −−lintian-opts −i

Note the order of options here: the **debuild** options come first, then the **dpkg-buildpackage** ones, then finally the checker options. (And **lintian** is called by default.) If you find yourself using the same **dpkg-buildpackage** options repeatedly, consider using the **DEBUILD_DPKG_BUILDPACKAGE_OPTS** configuration file option as described above.

To build a package for a sponsored upload, given *foobar_1.0-1.dsc* and the respective source files, run something like the following commands:

dpkg-source −x foobar_1.0-1.dsc
cd foobar-1.0
debuild −k0x12345678

where 0x12345678 is replaced by your GPG key ID or other key identifier such as your email address. Again, you could also use the **DEBUILD_DPKG_BUILDPACKAGE_OPTS** configuration file option as described above to avoid having to type the **−k** option each time you do a sponsored upload.

## SEE ALSO

**chmod**(1), **debsign**(1), **dpkg-buildpackage**(1), **dpkg-checkbuilddeps**(1), **fakeroot**(1), **lintian**(1), **su**(1), **sudo**(1), **super**(1), **devscripts.conf**(5), **dpkg-statoverride**(8)

## AUTHOR

The original **debuild** program was written by Christoph Lameter <clameter@debian.org>. The current version has been written by Julian Gilbey <jdg@debian.org>.