Deb-o-Matic Documentation

Release 0.22

Luca Falavigna

November 08, 2016

Contents

CHAPTER 1

What is Deb-o-Matic?

Deb-o-Matic is an easy to use utility to build Debian source packages, meant to help developers to automate the building of their packages with a tool that requires limited user interaction and a simple configuration.

It provides some useful features such as automatic chroot creation, rebuild of source packages, post-build checks, and much more. It is also extendible with modules that are loaded and executed during the build phases.

1.1 Why Deb-o-Matic?

When the author started to contribute to the Debian and Ubuntu development, he was running a 10-year-old PC and had a poor network connectivity. Downloading lots of packages had always been a nightmare, Canonical's PPAs were always busy compiling other packages because of the limited resources invested at the time, and wanna-build was (and still is) too complex to set up for relatively simple workflows.

A brand new software was created to help building source packages to avoid the burden of the compilation, without wasting too much time configuring complex softwares to work. Deb-o-Matic was born! A group of Debian and Ubuntu developers started to use it as their primary build machine to avoid playing with sbuild and long builds. Some of them still use Deb-o-Matic to build their packages.

Over time, Deb-o-Matic has been used by some FLOSS projects too. For example, Scilab Enterprises uses Deb-o-Matic to build Scilab in a transparent and automatic way. Every 5 minutes, a cronjob checks if any new commit happened and start a built process through Deb-o-Matic.

CHAPTER 2

Requisites

2.1 Build and installation requirements

In order to build and install Deb-o-Matic, the following packages are required:

- python3 (>= 3.2)

The following packages are required to install documentation and translations:

- python3-sphinx
- texlive-latex-base
- texlive-latex-recommended
- texlive-fonts-recommended
- texlive-latex-extra
- gettext

2.2 Runtime requirements

In order to be able to run Deb-o-Matic, the following packages are required:

- python3 (>= 3.2)
- sbuild (>= 0.67.0-1)
- schroot (>= 1.6.10-2)
- debootstrap, cdebootstrap, or qemu-debootstrap
- python3-toposort

To enable additional features, you may want to install these packages:

- python3-pyinotify (>= 0.8.6)
- gpgv
- debian-archive-keyring
- ubuntu-keyring
- lintian
- piuparts (>= 0.45)

- autopkgtest (>= 4.0)
- blhc
- devscripts
- apt-utils
- gnupg

An Internet connection is also required, broadband access is recommended because underlying programs will fetch a lot of megabytes from remote locations in order to create the chroots or satisfy build dependencies.

CHAPTER 3

Installation

3.1 Installation from Debian/Ubuntu repository

Deb-o-Matic is available in Debian and Ubuntu repositories, so you can launch the following command to install Deb-o-Matic on your system:

sudo apt-get install debomatic

Depending on your Debian or Ubuntu release, you could not have the latest version. It is usually suggested to use the latest one to obtain new features and bugfixes. You may want to check Deb-o-Matic home page from time to time to see whether a new release is available.

3.2 Installation of the release tarball

If the version in Debian/Ubuntu repositories is not the one you are looking for, you can download a release tarball from Deb-o-Matic download page.

Unpack the release tarball, enter the new created directory, and launch the following command:

sudo python3 setup.py install

If everything went smoothly, you should be able to launch debomatic from your terminal's command line.

3.3 Installation of Git snapshots

If you want to test bleeding-edge features, or want to be always up-to-date with the latest and greatest Deb-o-Matic versions, you can download development snapshots directly from upstream Git repository. Open a terminal and launch the following command:

git clone https://github.com/debomatic/debomatic.git

You can follow the instructions described in the *previous section* to install Deb-o-Matic on your system.

CHAPTER 4

Configuration

4.1 Configuration file

Deb-o-Matic configuration file is the most important file within Deb-o-Matic infrastructure. It defines almost every option needed by Deb-o-Matic. It is divided into five sections: `debomatic`, which contains the general options needed for Deb-o-Matic to work; `distributions`, which contains all the details related to the distributions Deb-o-Matic can build package for; `chroots`, which contains the options related to the creation of the chroots used to build packages; `gpg`, which contains the options related to GPG signature checking; `modules`, which contains the options related to module handling.

Other sections are optionally defined by each single module, their details will be discussed in the *Modules section*.

Caution: Configuration file must be formatted with Python ConfigParser syntax.

4.1.1 debomatic section

These options are mandatory, Deb-o-Matic refuses to start if one of these options is missing from the configuration file. Also, Deb-o-Matic needs to be restarted to pick any change to one of these options.

- `incoming`

  This option indicates the directory where Deb-o-Matic expects to find source packages to build, and in which it will store chroots, build logs, and resulting packages.

  Suggested value: `/incoming`

Caution: Make sure chosen directory exists before launching Deb-o-Matic, otherwise it will refuse to start.

- `debootstrap`

  This option indicates which debootstrap tool is required to create the chroot to build source packags from. Deb-o-Matic currently supports debootstrap, cdebootstrap, and qemu-debootstrap.

  Suggested value: `debootstrap`

Caution: Make sure chosen debootstrap utility is installed on your system, otherwise Deb-o-Matic will not be able to create chroots and build packages.

- `architecture`

This option indicates the architecture to build package for. To build packages for the same architecture of the running system, `system` can be used instead of specifying the exact one.

Suggested value: `system`

- `threads`

  This option indicates the maximum concurrent builds that can be executed. If more build tasks are scheduled, they will be blocked until a slot becomes available again. More concurrent builds require more CPU cycles, so you may want to try different configurations to fit your needs.

  `threads` takes an integer as parameter.

  Suggested value: `3`

- `inotify`

  This option indicates whether to enable inotify support, or not. inotify will notify Deb-o-Matic about the availability of a new task, and will immediately start a new thread according to the requested task.

  If `python3-pyinotify` is not available, Deb-o-Matic will fall back to a timer-based method, where new tasks will be analyzed periodically.

  `inotify` takes True or False as parameter.

  Suggested value: `True`

- `sleep`

  This option indicates the number of seconds between two consecutive checks for new packages or commands to process. This option is only useful if inotify support is disabled, or is not available.

  `sleep` takes an integer as parameter.

  Suggested value: `60`

- `logfile`

  This option indicates which file will be used to store information and warning messages issued by Deb-o-Matic during its execution.

  Suggested value: `/var/log/debomatic.log`

- `loglevel`

  This option indicates which kind of debug messages will be displayed. There are four levels: `error`, `warning`, `info`, and `debug`.

  Suggested value: `info`

4.1.2 distributions section

These options are mandatory, Deb-o-Matic refuses to start if one of these options is missing from the configuration file. Also, Deb-o-Matic needs to be restarted to pick any change to one of these options.

- `list`

  This option indicates the path of the distribution configuration file. More on those files will be discussed in the *Distributions file section*.

  Suggested value: `/etc/debomatic/distributions`

> **Caution:** Make sure chosen directory exists before launching Deb-o-Matic, otherwise it will refuse to start.

- `blacklist`

This option indicates a list of distributions that are not allowed to accept new packages to build. Files targeted for a blacklisted distribution will be automatically deleted.

Option must define a space-separated distribution names matching the ones listed in the *Distributions file section*.

Suggested value: `(blank field)`

- `mapper`

  This option indicates a list of distributions that, even if they are not defined by a distribution file (see *Distributions file section*), can build packages on top of another distribution. This is particularly useful to indicate distribution aliases (such as `sid <=> unstable`) or subsets (such as `vivid-proposed => vivid`).

  Option must define a Python dictionary where keys are the distributions indicated by the packages, and values are the distributions on which build packages upon.

  Suggested value: `{'sid': 'unstable'}`

4.1.3 chroots section

These options are mandatory, Deb-o-Matic refuses to start if one of these options is missing from the configuration file. Also, Deb-o-Matic needs to be restarted to pick any change to one of these options.

- `profile`

  This option indicates which schroot profile the chroots must adhere to. Profile files must be stored under `/etc/schroot` directory.

  Suggested value: ‘‘debomatic‘

Caution: Make sure chosen profile exists before launching Deb-o-Matic, otherwise all chroots will not be created.

- `commands`

  This option indicates the directory where sbuild commands are stored. Commands are executable scripts which are processed by sbuild during various build phases. Please refer to the sbuild (1) man page for additional details.

  At the moment, Deb-o-Matic provides a script to disable Internet connection within the chroot on Linux systems to avoid accessing remote resources during the build phase, and another one to increase the speed of unpacking of the depencencies in the chroots.

  Suggested value: `/usr/share/debomatic/sbuildcommands`

Caution: This directory needs to be bind mounted in the chroot for the scripts to be launched correctly. It is possible to do so by adjusting the schroot profile linked to the chroots used by Deb-o-Matic.

4.1.4 crossbuild section

This section is optional, Deb-o-Matic will start normally if this section is missing in the configuration file. Also, Deb-o-Matic needs to be restarted to pick any change to one of these options.

- `crossbuild`

  This option indicates whether to enable cross-build support, or not.

- `hostarchitecture`

This option indicates which host architecture to use when building source packages.

Caution: The architecture must have cross-compilation at compiler lever, otherwise it will not be possible to cross-compile source packages.

4.1.5 gpg section

These options are mandatory, Deb-o-Matic refuses to start if one of these options is missing from the configuration file. Also, Deb-o-Matic needs to be restarted to pick any change to one of these options.

`gnupg` package is required for these options to be effective.

- `gpg`

  This option indicates whether to enable signature checking support, or not. If enabled, Deb-o-Matic will delete unsigned files and files with signatures not available in its keyring.

  `gpg` takes True or False as parameter.

  Suggested value: `False`

- `keyring`

  This option indicates the gnupg keyring file in which Deb-o-Matic will look for valid and trusted GPG keys.

  Suggested value: `/etc/debomatic/debomatic.gpg`

Caution: Make sure keyring file exists and is populated with trusted keys if GPG support is enabled, otherwise no tasks will be processed.

4.1.6 modules section

These options are mandatory, Deb-o-Matic refuses to start if one of these options is missing from the configuration file. Also, Deb-o-Matic needs to be restarted to pick any change to one of these options.

More on modules handling will be discussed in the *Modules section*.

- `modules`

  This option indicates whether to enable module loading, or not.

  `modules` takes True or False as parameter.

  Suggested value: `True`

- `path`

  This option indicates the directory where Deb-o-Matic expects to find modules.

  Suggested value: `/usr/share/debomatic/modules`

- `threads`

  This option indicates how many modules Deb-o-Matic should launch in parallel.

  `threads` takes an integer as parameter.

  Suggested value: `5`

- `blacklist`

This option indicates a list of modules that are not allowed to be executed during build process.

Option must define a space-separated module names matching the ones listed in the *Modules section*.

Suggested value: `AutoPkgTest Blhc Lintian Mailer Piuparts`

4.2 Distributions file

This file is populated by sections, each of them named after a distribution to build packages for. Every section can define five options.

- `suite`

  This option indicates the base suite to create the chroot for. Normally, it is equal to its distribution, but there are some exceptions (for instance, experimental's suite is unstable).

  This option is mandatory.

- `mirror`

  This option indicates the mirror site which contains the primary package archive of the distribution.

  This option is mandatory.

- `components`

  This option contains a space-delimited list of components to use.

  This option is mandatory.

- `extramirrors`

  This option indicates additional mirrors to add in the chroot. More than one additional mirror can be defined, separated by a newline.

  This option is optional.

- `extrapackages`

  This option contains a space-delimited list of additional packages to install in the chroot during its creation.

  This option is optional.

CHAPTER 5

Run Deb-o-Matic

5.1 Launch Deb-o-Matic

Deb-o-Matic needs root privileges to be executed, otherwise it refuses to start. In order to launch it, you can use the following command:

sudo debomatic -c debomatic.conf

with `debomatic.conf` being the configuration file as described in the *Configuration section*. Make sure this file exists, otherwise Deb-o-Matic will refuse to start.

5.1.1 Interactive mode

Deb-o-Matic will try to enter daemon mode automatically. If that is not possible, Deb-o-Matic will be executed in interactive mode, and will be bound to the shell that executed it, as a regular process.

It is also possible to force interactive mode by passing `-i` or `--interactive` option while invoking `debomatic` command:

sudo debomatic -c debomatic.conf -i

This is particularly useful for debugging purposes.

5.1.2 Oneshot mode

Deb-o-Matic will try to build all files found in the `incoming` directory. Alternatively, it is possible to attempt to build a sinle file by passing `-o` or `--oneshot` option while invoking `debomatic` command, following by the file name of the package to build, which must be located in the `incoming` directory:

sudo debomatic -c debomatic.conf -o package_version_source.changes

5.2 Stop Deb-o-Matic

In order to stop Deb-o-Matic, you should pass `-q` or `--quit` option to `debomatic`:

sudo debomatic -c debomatic.conf -q

Deb-o-Matic will not terminate child processes immediately, but will wait for them to end first, so it could take a while to completely stop a Deb-o-Matic instance.

Caution: Deb-o-Matic uses a rather strong locking mechanism, so it is not recommended to terminate debomatic process with `kill` command.

5.3 Using service command

If you installed Deb-o-Matic using Debian package, you could start, stop, and restart Deb-o-Matic with the following commands, respectively:

sudo service debomatic start

sudo service debomatic stop

sudo service debomatic restart

You will need to adjust configuration stored in `/etc/default/debomatic` file to manage Deb-o-Matic with this method, though. In particular, you will have to set `DEBOMATIC_AUTOSTART` variable to 1.

5.3.1 Service configuration

In order to start Deb-o-Matic with `service` command, you must adjust some parameters defined in `/etc/default/debomatic` file.

- `DEBOMATIC_AUTOSTART`

  This option indicates whether to execute Deb-o-Matic at system boot. Default value is set to `0` to avoid accidental executions without a sane configuration. It must be set to `1` in order to launch Deb-o-Matic.

- `DEBOMATIC_CONFIG_FILE`

  This option indicates the configuration file Deb-o-Matic is going to use.

- `DEBOMATIC_OPTS`

  This option allows to pass extra options to Deb-o-Matic.

5.4 Using systemctl command

If you installed Deb-o-Matic using Debian package, and your system does use of systemd as default init, you could start, stop, and restart Deb-o-Matic with the following commands, respectively:

sudo systemctl start debomatic

sudo systemctl stop debomatic

sudo systemctl restart debomatic

systemd unit file is configured to look for `/etc/debomatic/debomatic.conf` as its default configuration file. You can change this path by providing a systemd override file.

CHAPTER 6

Prepare source packages

Deb-o-Matic will take into account both source only uploads and source and binary uploads, while it will discard binary only uploads. Source only uploads are recommended to avoid waste of bandwidth, so make sure you create packages by passing `-S` flag to `debuild` or `dpkg-buildpackage`.

Then, packages must be copied or uploaded into the directory specified by the `incoming` option in the configuration file to let Deb-o-Matic process them.

In order to save bandwidth while uploading your packages, you could want to avoid including upstream tarball in the .changes file if it is already available in the distribution mirrors, Deb-o-Matic will fetch it automatically for you. In order to do so, you have to pass `-sd` flag to `debuild` or `dpkg-buildpackage`.

Multiple uploads of the same packages are allowed, Deb-o-Matic will overwrite previous builds with new, fresh files.

6.1 User-defined fields

sbuild uses several resolvers to determine and install build-dependencies inside the chroots. Sometimes it is desirable to override the default resolver to perform some advanced tasks (e.g. using a specific version of a package which apt-based resolver cannot pick automatically.

In order to do so, you must define the `XC-Debomatic-Resolver` in the source stanza of your `control file`. For instance, if you want to use the aptitude resolver, you must use the following syntax:

XC-Debomatic-Resolver: aptitude

CHAPTER 7

Prepare command files

Deb-o-Matic provides an interface to perform specific tasks into the Deb-o-Matic `incoming` directory such as removing uploaded files or rebuilding packages. These operations are handled by commands stored in `.commands` files, and uploaded into Deb-o-Matic `incoming` directory by using `dcut` utility, or by hand.

Using dcut is usually simpler, just launch the following command:

dcut -U mydebomatic commandfile.commands

where `mydebomatic` is a dput host as described in dput.cf (5) man page, and `commandfile.commands` is the file containing the commands to be executed by Deb-o-Matic.

Multiple commands can be stored in a single `.commands` file, but it is usually safer to issue a single command per file.

Caution: If signature checking support is enabled, .commands files must be signed by a known key, otherwise they will be deleted and no action will be taken.

7.1 Remove packages

It could happen some files are kept into Deb-o-Matic `incoming` directory, and you would like to remove them. In order to do so, you must use the `rm` command:

echo "rm foo" > foo.commands*

where `foo*` is a regular expression matching the files you want to remove.

7.2 Rebuild packages

You could want to rebuild a package already in the mirrors to see whether it compiles with newer packages, to analyze its content, and so on. In order to do so, you must use the `rebuild` command:

echo "rebuild foo_version dist" > foo.commands

where `foo` is the name of the source package you want to rebuild, `version` is the version of the package you want to rebuild, and `dist` is the distribution which rebuild the package for.

Deb-o-Matic can also rebuild packages available in other distributions. The syntax is similar, you just have to indicate which distribution to pick packages from:

echo "rebuild foo_version dist origin" > foo.commands

where `origin` is the distribution to pick packages from.

Caution: Make sure packages are available in the distribution mirrors, otherwise they cannot be downloaded and processed by Deb-o-Matic.

7.3 Porter uploads

You could want to prepare a porter upload, a binary-only upload which generates architecture dependent binaries only. Additional information can be found in Debian Developer's Reference.

In order to do so, you must use the `porter` command:

echo "porter foo_version dist John Doe <jdoe@debian.org>" > foo.commands

where foo is the name of the source package you want to rebuild, version is the version of the package you want to rebuild, dist is the distribution which rebuild package for, and the rest of the string is the address to be used as maintainer field, which is usually the developer who is preparing the upload.

Caution: Make sure packages are available in the distribution mirrors, otherwise they cannot be downloaded and processed by Deb-o-Matic.

7.4 Binary NMU uploads

You could want to prepare a binary NMU (or binNMU) upload, a binary-only upload which generates architecture dependent binaries only, together with a changelog entry describing why the upload was needed. Additional information can be found in Debian Developer's Reference.

In order to do so, you must use the `binnmu` command:

***echo "binnmu foo_version dist binNMU_version "changelog"** John Doe <jdoe@debian.org>" > foo.commands*

where foo is the name of the source package you want to rebuild, version is the version of the package you want to rebuild, dist is the distribution which rebuild package for, binNMU_version is the progressive binNMU number, changelog is the reason why the upload was prepared (enclosed in quotation marks), and the rest of the string is the address to be used as maintainer field, which is usually the developer who is preparing the upload.

Caution: Make sure packages are available in the distribution mirrors, otherwise they cannot be downloaded and processed by Deb-o-Matic.

7.5 Rebuild packages with extra build-dependencies

You could want to rebuild a package already in the mirrors also adding a specific build-dependency to see whether it compiles with a newer library version. In order to do so, you must use the `builddep` command:

echo "builddep foo_version dist extrapackage (>= packageversion)" > foo.commands

where `extrapackage` is the name of the package you want to install before the compilation takes place, and `packageversion` is the optional version of the package you want to install. More than one package can be defined, separated by commas.

Caution: Make sure packages are available in the distribution mirrors, otherwise they cannot be downloaded and processed by Deb-o-Matic.

7.6 Killing builds

You could want to terminate a build you erroneously uploaded, or you do not want it to complete to avoid wasting too many resources.

In order to do so, you must use the `kill` command:

echo "kill foo_version dist " > foo.commands

where foo is the name of the source package you want to terminate its build, version is its versiond, and dist is the distribution the package is being built for.

CHAPTER 8

Modules

8.1 Autopkgtest

This module allows adt-run to be executed if source package declares a Testsuite against autopkgtest. It creates a report in the same directory of the resulting files.

8.1.1 Parameters

Caution: These parameters must be listed under the `autopkgtest` section. Make sure you create it in your configuration file.

- `options`

This option indicates the extra options to pass to adt-run.

Suggested value: `--no-built-binaries`

8.2 Blhc

This module allows blhc to be executed, checking the build log of built packages for missing hardening flags.

In order for this module to work properly, `blhc` package must be installed.

8.3 Parameters

Caution: These parameters must be listed under the `blhc` section. Make sure you create it in your configuration file.

- `options`

This option indicates the extra options to pass to blhc.

Suggested value: `--all`

8.4 Contents

This module scans binary packages and stores their content in a `.contents` file created in the same directory of the resulting files.

In order for this module to work properly, `debc` tool from `devscripts` must be available.

8.5 DateStamp

This module displays timestamps of when a package started to build, when it finished, and the build elapsed time. Timestamps are stored in a `.datestamp` file created in the same directory of the resultinf files.

8.6 Lintian

This module allows lintian to be executed, checking the built packages for errors and warnings, and creates a report in the same directory of the resulting files.

In order for this module to work properly, `lintian` package must be installed.

8.6.1 Parameters

Caution: These parameters must be listed under the `lintian` section. Make sure you create it in your configuration file.

- `options`

This option indicates the extra options to pass to lintian.

Suggested value: `-iIE --pedantic`

8.7 Mailer

This module allows to send emails about the status of the builds. Body of the email will contain an excerpt of the build log to easily see failures or potential problems.

Caution: Make sure signature checking support is enabled before trying to use this module, otherwise it will not work as it relies on the address provided in the GPG key to obtain the email address to send messages to.

8.7.1 Parameters

Caution: These parameters must be listed under the `mailer` section. Make sure you create it in your configuration file.

- `sender`

This option indicates the email address used to send the emails from.

- `server`

This option indicates the SMTP server used to send the emails.

- `port`

This option indicates the SMTP port on which the SMTP server listens to.

- `tls`

This option indicates whether to enable TLS mode, or not.

- `authrequired`

This option indicates whether the SMTP server requires authentication, or not.

- `user`

This option indicates the user name to be passed to the SMTP server.

- `pass`

This option indicates the password to be passed to the SMTP server.

- `success`

This option indicates the template to be used to report successful builds.

- `failure`

This option indicates the template to be used to report failed builds.

- `lintian`

This option indicates whether the lintian log is to be attached after the build log, or not.

8.8 Piuparts

This module allows piuparts to be executed, checking the built packages for potential problems, and creates a report in the same directory of the resulting files.

In order for this module to work properly, `piuparts` package must be installed.

8.8.1 Parameters

> **Caution:** These parameters must be listed under the `piuparts` section. Make sure you create it in your configuration file.

- `options`

This option indicates the extra options to pass to piuparts.

Suggested value: `--log-level=info`

8.9 PrevBuildCleaner

This modules deletes obsolete files created during previous builds to avoid picking obsolete files by mistake. It currently deletes these files:

- *.deb
- *.ddeb

- *.gz
- *.bz2
- *.xz
- *.dsc
- *.build
- *.contents
- *.lintian
- *.piuparts
- *.changes
- *.autopkgtest
- *.bhlc

8.10 Repository

This module allows the creation of a simple repository of Debian binary packages, which is refreshed each time a build is performed, allowing to build packages build-depending on previously built ones. In order for this module to work properly, `apt-ftparchive` tool from `apt-utils` package must be available.

8.10.1 Parameters

Caution: These parameters must be listed under the `repository` section. Make sure you create it in your configuration file.

- `gpgkey`

This option indicates the GPG ID used to sign the Release file of the repository.

- `pubring`

This option indicates the path where to look for the public GPG key used to sign the Release file of the repository.

- `secring`

This option indicates the path where to look for the private GPG key used to sign the Release file of the repository.

8.11 SourceUpload

This module allows the creation of a .sourceupload.changes file to be used to upload source-only uploads to the Debian archive.