g-Octave Documentation

Release 0.4.1

Rafael Goncalves Martins

December 22, 2015

Contents

Author Rafael Goncalves Martins

Website http://www.g-octave.org/

Source code https://github.com/rafaelmartins/g-octave

Bugs to https://github.com/rafaelmartins/g-octave/issues

Version 0.4.1

Overview

g-Octave is a tool that generates and installs ebuilds for Octave-Forge packages "on-the-fly" to Gentoo Linux, using Portage, Paludis or Pkgcore. It's capable to generate ebuilds and Manifest files (if needed) for the packages, and to install them using an autogenerated overlay (named g-octave). g-Octave can also handle patches to the packages automatically. The command line interface tries to be very similar to the interface of the `emerge` tool.

CHAPTER 1

Contents

1.1 Quick Start

This is a svery small set of instructions for the users that just want to install a package, and don't care about things like which package manager g-octave will use or where g-octave will store your files.

1.1.1 Installing g-Octave

With layman installed, install the science overlay:

```
# layman -a science
```

Install g-octave:

```
# emerge -av g-octave
```

Install g-octave's package database:

```
# emerge --config g-octave
```

1.1.2 Installing the package

If you don't care about configurations, the default values are good enough for you, then just type:

```
# g-octave -av packagename
```

1.1.3 Just a line!

If you're really lazy, you can just type (first time):

```
# layman -a science && emerge g-octave && emerge --config g-octave && g-octave packagename
```

and when you want to install other package, you can type:

```
# g-octave otherpackagename
```

1.2 User Guide

This is an user guide with some instructions to the end-user.

1.2.1 Installing g-Octave

The ebuilds for g-Octave are available on the Portage tree. You can install the package, using:

```
# emerge -av app-portage/g-octave
```

We have 2 ebuilds, one for with latest stable release (for `~x86` and `~amd64`) and one live ebuild, that installs g-Octave from the Git repository (without keywords). If you want to use the live ebuild, you need to unmask it adding the line below to your `/etc/portage/package.keywords`:

```
app-portage/g-octave **
```

The live ebuild is only recommended for who want to help testing new features, or for developers.

Stable users (with `x86` or `amd64`) that wants to test the latest release will need to unmask the ebuild too, adding this to /etc/portage/package.keywords (e.g. for `x86`):

```
app-portage/g-octave ~x86
```

The source code of g-Octave can be found in this Git repository:

https://github.com/rafaelmartins/g-octave/downloads

You can clone the Git repository using this command (with Git installed, of course):

```
$ git clone git://github.com/rafaelmartins/g-octave.git
```

The release tarballs can be found here:

https://github.com/rafaelmartins/g-octave/downloads

USE flags

- `doc`: Install this documentation. Depends on `dev-python/sphinx`.

1.2.2 Configuring g-Octave

Using the file `/etc/g-octave.cfg`

If you installed g-Octave correctly, you should find a configuration file at `/etc/g-octave.cfg`.

The main options are `package_manager`, `db` and `overlay`, that defines the package manager used by g-octave and the directory paths for the package database and the generated overlay, respectively.

Other options are available. Please read the comments in the configuration file.

Using environment variables

All the options from the configuration file can be overrided with environment variables. The environment variable name starts with `GOCTAVE_` and ends with the option name in uppercase. for example, `GOCTAVE_OVERLAY` will override the option `overlay` from the config file.

Usage example:

```
# GOCTAVE_OVERLAY=/tmp/overlay g-octave -av packagename
```

1.2.3 Enabling the logging feature

If you want to write some relevant stuff to a log file you can enable the logging feature, configuring the option `log_level` on the configuration.

The available options are: `debug`, `info`, `warning`, `error`, `critical`.

You can change the location of the log file, using the option `log_file`. The default is: `/var/log/g-octave.log`

Make sure that the user running g-octave have write permissions to `log_file`.

1.2.4 Syncronizing the package database

Currently g-Octave depends on an external package database, in order to create the ebuilds for the packages. If you installed the live version of g-Octave (=g-octave-9999) you'll need to fetch this database in the first time that you run g-Octave (and whenever you want to updates):

```
# g-octave --sync
```

1.2.5 Configuring your package manager

g-octave can use all the 3 package managers available on Gentoo Linux: **Portage**, **Paludis** and **Pkgcore**.

You just need to setup the option `package_manager` with the lowercase name of the package manager: `portage`, `paludis`, `pkgcore`.

If you're using **Paludis** or **Pkgcore**, you'll need to configure the overlay in your package manager configuration files. Please check the documentation of your package manager:

- Paludis: http://paludis.pioto.org/
- Pkgcore: http://www.pkgcore.org/

Portage works out of the box.

1.2.6 Installing packages

From the upstream source tarballs

You can list all the available packages using this command:

```
# g-octave --list
```

or

```
# g-octave -l
```

To install a package, use:

```
# g-octave packagename
```

or

```
# g-octave packagename-version
```

For example:

```
# g-octave control-1.0.11
```

`g-octave` command-line tool supports some options for the installation of packages:

`-a` or `--ask` Ask before install the package

`-p` or `--pretend` Only pretend the installation of the package

`-1` or `--oneshot` Do not add the packages to the world file for later updating.

You can get some information about the package using this command:

```
# g-octave --info packagename
```

or

```
# g-octave -i packagename
```

From the octave-forge Mercurial repository

If you want to test some new feature or to always use the newest version of the packages, you'll like to install the packages directly from the Mercurial repository.

To install a package from Mercurial, you'll need to configure g-Octave, changing the value of the variable `use_scm` on the file `/etc/g-octave.cfg` to `true`. After that, type:

```
# g-octave packagename
```

If you only want to install a single package, you can use the command-line option `--scm`.

If you enabled the installation from Mercurial on the configuration file and wants to install a stable version, you can use the command-line option `--no-scm`.

1.2.7 Updating packages

You can update a package using this command:

```
# g-octave --update packagename
```

or

```
# g-octave -u packagename
```

If you want to update all the installed packages, run this without arguments:

```
# g-octave --update
```

or

```
# g-octave -u
```

The options `--ask` and `--verbose` are also supported.

1.2.8 Searching packages

You can do searches on the package names if you use the option `-s` or `--search`. Regular expressions are allowed.

```
# g-octave --search anything
```

or

```
# g-octave -s ^con
```

1.2.9 Uninstalling packages

You can uninstall packages using this command:

```
# g-octave --unmerge packagename
```

or

```
# g-octave -C packagename-version
```

The options `--ask` and `--verbose` are also supported.

1.2.10 Troubleshooting

Some times the generated ebuilds can be broken for some reason. To fix this you can use the command-line option `--force`, that will rebuild the ebuild or the command-line option `--force-all`, that rebuild the entire overlay.

If you got some problem with corrupted sources, please remove the tarball from the `${DISTDIR}` and run:

```
# g-octave --force packagename
```

If you still have problems, please fill a ticket on our bug tracker

1.3 Upgrading g-Octave

Quick notes for users upgrading from older versions of g-Octave.

1.3.1 From `0.3` to `0.4`

This version broke the compatibility with old package databases and some external files.

You'll need to remove the directories of the package database and the overlay (`db` and `overlay` options from the configuration file).

To know what are the current values, type:

```
$ g-octave --config db
$ g-octave --config overlay
```

And remove both directories before run `emerge --config app-portage/g-octave`

1.4 Development

Source code https://github.com/rafaelmartins/g-octave

Bugs to https://github.com/rafaelmartins/g-octave/issues

Warning: This section of the documentation is supposed to be used by g-Octave developers. End-users should not need to read this!

1.4.1 Running the test suites

You can run the tests suites using the script `run_tests.py` that can be found in the directory `scripts` in the recent source tarballs or in the Git repository

```
$ scripts/run_tests.py
```

If some test is broken, please use the bug tracker.

1.4.2 Working on the package database

Package databases are Git repositories with the DESCRIPTION files, patches and a `info.json` file with the non-octave dependencies and the licenses of the packages.

We're using github to host the package database:

http://github.com/rafaelmartins/g-octave-db/

If you want to fix something on the package database, please fork the repository, change it and fill a merge request.

Updating the package database

We have a script to update the package database: `contrib/manage_pkgdb.py`. You just need to clone the git repository and create a directory to store the source tarballs, that are used to extract the DESCRIPTION files and build the package database.

g-Octave will install the script on `/usr/share/g-octave/contrib`

```
$ git clone git+ssh://git@github.com:your_username/g-octave-db.git
$ mkdir tarballs
$ /usr/share/g-octave/contrib/manage_pkgdb.py tarballs g-octave-db
```

The first parameter of the script is the path to the directory that will store the tarballs. The second parameter is the path to the local clone of your forked git repository.

You'll need an additional dependency:

```
# emerge -av dev-python/feedparser
```

Updating the list of external dependencies and licenses

We have a script to update the JSON file that contains the external dependencies (non-octave packages from the portage tree) and the licenses (the license names used by the octave-forge developers doesn't matches with the names used on the Portage tree).

This script will be also installed on `/usr/share/g-octave/contrib`

```
$ /usr/share/g-octave/contrib/manage_info.py g-octave-db/info.json
```

The script is interactive and the argument is the path to the `info.json` file, that lives on the root of the Git repository of the package database.

The script will suggest some matches for each dependency. For the licenses you need to find the best match at the directory `${PORTDIR}/licenses`, where ${PORTDIR} is the path to your Portage tree (`/usr/portage` usually).

Commiting the changes

You can use the script `manage_pkgdb.py` to commit the changes:

```
$ /usr/share/g-octave/contrib/manage_pkgdb.py --commit tarballs g-octave-db
```

The script will do a last check on your updates and commit the stuff to your fork repository.

Using your fork as package database

If you want to use your fork as package database, change the variable `db_mirror` on the file `/etc/g-octave.cfg` to something like:

```
db_mirror = github://your_username/g-octave-db
```

1.4.3 Sending patches to the source code

The source code of g-Octave lives on a repository on the Gentoo Linux infrastructure.

```
$ git clone git://github.com/rafaelmartins/g-octave.git
```

You can change what you need, commit, generate a Git-formated patch and attach it to a new ticket on our bug tracker.