

# How to Create DEB Packages for Debian/Ubuntu

 [makeuseof.com/create-deb-packages-debian-ubuntu](https://makeuseof.com/create-deb-packages-debian-ubuntu)

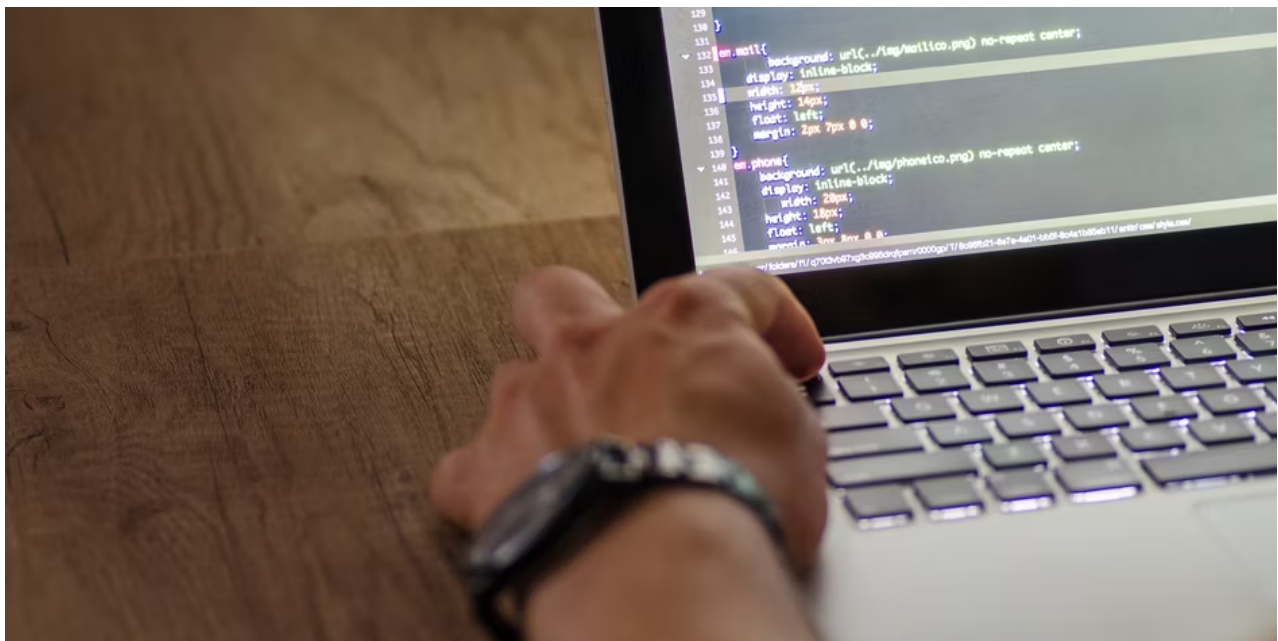
June 8, 2022

- [Home](#)
- [Linux](#)

By [Fatih Küçükçarakurt](#)

Published Jun 08, 2022

Most of the software on Linux is distributed as packages. Creating DEB packages for Debian-based distros is easier than you might think.



Readers like you help support MUO. When you make a purchase using links on our site, we may earn an affiliate commission.

A DEB package is an archive containing all the files including the compiled version of the applications, source codes, configuration files, images, and installation commands. DEB files in Debian-based operating systems like Ubuntu and Kali Linux are equivalent to the EXE files found in Windows.

Here's how you can develop your own DEB packages for a Debian-based Linux distro.

102.8K

GE Cync Smart Lighting: Don't Buy Philips Hue Until You've Seen These!

[  
primis  
]



Next

Stay

## Step 1: Installation of Required Packages

---

Preparing a Debian package requires some programs. To begin, install these utilities on your system:

```
sudo apt install build-essential binutils lintian debhelper dh-make devscripts
```

## Step 2: Package Selection

---

Before creating a Debian package (DEB) for a program, you should consider a few points:

- Check whether the package you're planning to create is already in the Debian repositories:

```
apt-cache search package-name
```

- Check what kind of license the program you are going to package has. There is a general culture of using the GNU/GPL license.
- Make sure that the program does not pose a security problem for the system.
- Contact the author of the program. Notify Debian developers for this program to enter the Debian repositories.

## Step 3: Begin to Prepare the Package

---

Firstly, create a new directory under your home directory to avoid confusion.

```
cd /home  
mkdir package  
cd package
```

Then extract the tar archive containing the source code of the program you're going to package under this directory. For demonstration, we'll use the rsyslog archive.

```
tar -zxvf rsyslog-6.3.6.tar.gz
```

Navigate to the newly-created directory using the cd command:

```
cd rsyslog-6.3.6
```

Usually, the source code of the program comes with **INSTALL** and **README** files. Even if you know what the program is and how it works, it will be beneficial for you to spend some time reading these files.

There are commands such as **./configure make** and **make install** that can easily install such archives on your system. But there are several parameters for the **./configure** option that you should be aware of. You can use the **./configure --help** command to get such information.

## Step 4: Adding Developer Information

---

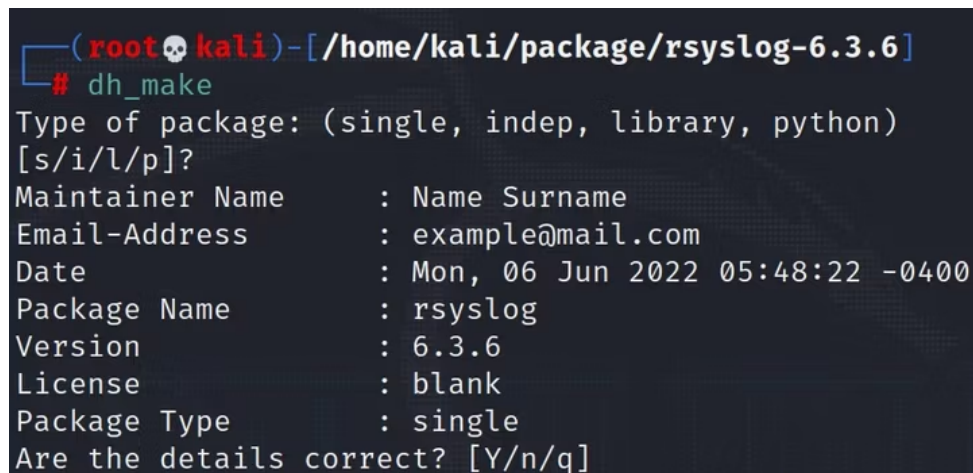
Before creating a DEB package for your program, pay attention to the package name and version number. You will also need to add some packager information when creating a package. For this, you must export your information with the following commands:

```
export DEBEMAIL="your@mail.com"
export DEBFULLNAME="Name Lastname"
```

After this, use the magic command **dh\_make**.

```
dh_make
```

After issuing the **dh\_make** command, you should select your package type and press **Enter**.



```
(root@kali)-[/home/kali/package/rsyslog-6.3.6]
# dh_make
Type of package: (single, indep, library, python)
[s/i/l/p]?
Maintainer Name      : Name Surname
Email-Address        : example@mail.com
Date                 : Mon, 06 Jun 2022 05:48:22 -0400
Package Name         : rsyslog
Version              : 6.3.6
License              : blank
Package Type         : single
Are the details correct? [Y/n/q]
```

Following this step, you will notice a directory in a parent directory with the ".orig" extension. If this doesn't work, try running the **dh\_make** command with the **--createorig** parameter.

```
ls
# Output
rsyslog-6.3.6  rsyslog_6.3.6.orig.tar.xz  rsyslog-6.3.6.tar.gz
```

You can also see a new directory named **Debian** in the present working directory. These directories and files hold all the Debian package-related information about the program.

```
(root@kali)-[/home/kali/package/rsyslog-6.3.6]
# cd debian

(root@kali)-[/home/kali/package/rsyslog-6.3.6/debian]
# ls
changelog      manpage.sgml.ex  prerm.ex        rsyslog-docs.docs
control        manpage.xml.ex   README.Debian    rules
copyright      postinst.ex      README.source    salsa-ci.yml.ex
manpage.1.ex   postrm.ex        rsyslog.cron.d.ex source
manpage.md.ex  preinst.ex       rsyslog.doc-base.ex watch.ex
```

## Step 5: Package-Related Debian Files

You need to know the following information about the files located in the Debian directory.

### 1. The control File

The control file offers a variety of package-related information.

```
(root@kali)-[/home/kali/package/rsyslog-6.3.6/debian]
# cat control
Source: rsyslog
Section: unknown
Priority: optional
Maintainer: Name Surname <example@mail.com>
Build-Depends: debhelper-compat (= 13), autotools-dev
Standards-Version: 4.6.0
Homepage: <insert the upstream URL, if relevant>
#Vcs-Browser: https://salsa.debian.org/debian/rsyslog
#Vcs-Git: https://salsa.debian.org/debian/rsyslog.git
Rules-Requires-Root: no

Package: rsyslog
Architecture: any
Depends: ${shlibs:Depends}, ${misc:Depends}
Description: <insert up to 60 chars description>
             <insert long description, indented with spaces>
```

- **Source:** The line where you will specify the name of your program
- **Section:** The line that determines which section your program belongs to according to the license
- **Maintainer:** The line containing the information of the person who prepared the package
- **Build-Depends:** Dependencies are listed on this line
- **Depends:** This line is very important. You specify the dependencies of your package with this value
- **Description:** The line where you can enter information about the package

### 2. The copyright File

---

This file contains information about the license of the program. Its default content is as follows:

```
(root@kali)-[/home/kali/package/rsyslog-6.3.6/debian]
# cat copyright
Format: https://www.debian.org/doc/packaging-manuals/copyright-format/1.0/
Upstream-Name: rsyslog
Upstream-Contact: <preferred name and address to reach the upstream project>
Source: <url://example.com>

Files: *
Copyright: <years> <put author's name and email here>
          <years> <likewise for another author>
License: <special license>
        <Put the license of the package here indented by 1 space>
        <This follows the format of Description: lines in control file>
        .
        <Including paragraphs>

# If you want to use GPL v2 or later for the /debian/* files use
# the following clauses, or change it to suit. Delete these two lines
Files: debian/*
Copyright: 2022 Name Surname <example@mail.com>
License: GPL-2+
    This package is free software; you can redistribute it and/or modify
    it under the terms of the GNU General Public License as published by
    the Free Software Foundation; either version 2 of the License, or
    (at your option) any later version.
    .
    This package is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
```

### 3. The changelog File

---

This file is like your program's logbook roadmap. If you have done something independent of the source of the program or if you have fixed some bugs, you can add it to this file.

### 4. The rules File

---

The rules file is like a Makefile for your Debian package. When installing the prepared Debian package with dpkg, the information in this file is taken as a base.



```
(root@kali)-[/home/kali/package/rsyslog-6.3.6/debian]
# cat rules
#!/usr/bin/make -f
# See debhelper(7) (uncomment to enable)
# output every command that modifies files on the build system.
#export DH_VERBOSE = 1

# see FEATURE AREAS in dpkg-buildflags(1)
#export DEB_BUILD_MAINT_OPTIONS = hardening=+all

# see ENVIRONMENT in dpkg-buildflags(1)
# package maintainers to append CFLAGS
#export DEB_CFLAGS_MAINT_APPEND = -Wall -pedantic
# package maintainers to append LDFLAGS
#export DEB_LDFLAGS_MAINT_APPEND = -Wl,--as-needed

%:
    dh $@

# dh_make generated override targets
# This is example for Cmake (See https://bugs.debian.org/641051 )
#override_dh_auto_configure:
#    dh_auto_configure -- \
#    -DCMAKE_LIBRARY_PATH=$(DEB_HOST_MULTIARCH)
```

You can of course change the parameters in this file as you see fit.

## 5. Other Files in the Directory

---

It may be useful to know the functions of the following files too:

- **README.Debian:** Readme file
- **conffiles.ex:** Use this file if you want to keep your old settings file while installing the program
- **cron.d.ex:** You can perform cron operations using this file
- **dirs:** Use this file to specify directories that should not be installed during the installation but should be created later
- **docs:** If there are documents with your program, specify them with this file
- **emacsen\*.ex:** If your program needs the Emacs file during installation, specify it with this file
- **init.d.ex:** Use this file if you want your program to run at system startup

To proceed to the following stage, remove any files you believe you no longer require. Then rename the file extensions and remove ".ex" from the end. The ".ex" (example) indicates that this is an example file.

## Step 6: Building the Package

---

If you have come this far, you can now prepare the Debian package for your program. For this, run the following command:

```
dpkg-buildpackage
```

Another important issue here is to create a GPG for the e-mail address you export as Maintainer.

```
export DEBEMAIL="example@mail.com"
```

dpkg will look for your GPG information while creating the package. You can list it with the command **gpg --list-keys**.

If you encounter any problems in the **dpkg-buildpackage** phase, try the following command:

```
dpkg-buildpackage -nc -i
```

This command will ignore some parts that may cause an error.

If everything goes well, the Debian package for your program will be ready to install and stored in the next directory. With the command below, you can install, test, and review the package.

```
dpkg -i package-name
```

```
(root@kali)-[/home/kali/package]
# dpkg -i rsyslog 6.3.6-1 amd64.deb
dpkg: warning: downgrading rsyslog from 8.2202.0-1 to 6.3.6-1
(Reading database ... 325557 files and directories currently installed.)
Preparing to unpack rsyslog_6.3.6-1_amd64.deb ...
Unpacking rsyslog (6.3.6-1) over (8.2202.0-1) ...
Setting up rsyslog (6.3.6-1) ...
Processing triggers for man-db (2.10.2-1) ...
Processing triggers for kali-menu (2021.4.2) ...
```

## Anyone Can Create a Package on Linux

---

The DEB packaging system is one of the most fundamental elements that distinguishes Debian as a GNU/Linux leader. Debian is a large system and it's really important for contributors to have the ability to create their own packages.

If you're new to GNU/Linux, this may seem perplexing. However, as you can see, preparing a Debian package is simpler than you may think. Of course, building a Debian package requires time and work.

But that doesn't mean you have to manually create packages for programs that you want to install. There are several websites on the internet from where you can download DEB packages for free.

## Poll

---

**Would you pay for verification on Twitter, and if so, how much?**

---

