

# Packaging a C/C++ Application into a .deb package for Debian/Ubuntu: An Easy Guide for Beginners

The world of Linux offers a plethora of opportunities for software developers, one of which is the ability to package applications into .deb files. This format is widely used in Debian-based systems like Ubuntu, Linux Mint, PopOS, and others, making software distribution more manageable and user-friendly. In this guide, we will walk you through the process of creating a .deb package for a simple C/C++ application, demystifying the packaging process.



Jan Bronicki · [Follow](#)

Published in CodeCurrent · 5 min read · Feb 3, 2024



9





## Starting with a Basic C++ Program

The journey begins by writing a simple “Hello World” program in C++. This step is crucial as it provides us with a working binary to package. After writing the program, it’s compiled using `g++`. This process results in a binary file that, while functional, isn’t yet in a distributable format. For this reason let’s create a simple, sample “Hello World” program in C++:

```
#include <iostream>

int main()
{
    std::cout << "Hello world!" << std::endl;
}
```

Now let's build it and run it so that we can be sure that it works:

```
$ g++ ./main.cpp -o myapp
$ ./myapp
Hello world!
```

Before diving into the packaging process, it's essential to ensure that you can access Debian packaging-specific commands. If you are not on a Debian-based system you can simply open your project in a docker container and build it there:

```
docker run -it -v <path-to-your-project>:<where-to-mount-in-docker> debian:lates
```

This is the exact command I ran, it mounts my project directory in /app directory of the docker container:

```
docker run -it -v ~/Documents/Programming/debpkging/my-app:/app debian:latest
```

## Setting Up the Directory Structure

The key to Debian packaging lies in creating a specific directory structure. This structure mimics the file system of a Debian-based machine, providing a blueprint for where the files should reside upon installation. Initially,

[Open in app](#) ↗

[Sign up](#)

[Sign in](#)

Medium

 Search

 Write



```
.
|-- main.cpp
`-- myapp/
    |-- DEBIAN/
        |-- control
```

## Crafting the Control File

Inside the `DEBIAN/` folder, you create a control file. This file is vital as it contains essential metadata about your package, such as the package name, version, and a brief description. It's a straightforward file but is crucial for the Debian packaging system to recognize and manage your package correctly.

```
Package: myapp
Version: 0.1
Maintainer: Jan Bronicki
Architecture: arm64
Description: My App
```

## Placing the Binary

Once the control file is set up, the next step is to place the compiled binary in the correct location within your package directory. If your application is a system-level tool, it typically goes into a folder structure like `usr/bin/` within your package directory. However, in the configuration, you don't include the 'root' ( `/` ) part of the path. This might seem confusing at first, but it's a crucial step in ensuring your binary is correctly located. Remember, this structure represents where the files will go on the user's system, not your file system:

```
$ g++ ./main.cpp -o ./myapp/usr/bin/myapp # We can simply output the compiled bi
$ tree
.
|-- main.cpp
`-- myapp/
    |-- DEBIAN/
    |   `-- control
    `-- usr/
        `-- bin/
            `-- myapp
```

Here I would like to mention that where you should put your binaries has some logic behind it, for the sake of the simplicity of this guide we are going to put them in `/usr/bin/` but you should research beforehand where your binaries should be put on a production system, here is a great [article that outlines that](#).

## Building the .deb Package

With the directory structure and control file in place, you're ready to build the `.deb` package. This is done using the `dpkg-deb --build` command, which bundles everything into a single `.deb` file. This file is what you will distribute to users.

```
$ dpkg-deb --build ./myapp
dpkg-deb: building package 'myapp' in './myapp.deb'.
```

Now you should be able to see your Debian package in the root of the repository:

```
.
|-- main.cpp
|-- myapp/
|   |-- DEBIAN/
|   |   `-- control
|   `-- usr/
|       `-- bin/
|           `-- myapp
`-- myapp.deb
```

## Installing and Testing the Package

After building the package, you can install it using the `dpkg -i` command followed by the path to your `.deb` file:

```
$ dpkg -i ./myapp.deb
Selecting previously unselected package myapp.
(Reading database ... 16606 files and directories currently installed.)
Preparing to unpack ./myapp.deb ...
Unpacking myapp (0.1) ...
Setting up myapp (0.1) ...
```

Once installed, you can run the application like any other system command:

```
$ myapp
Hello world!
```

It's also a good practice to check that your application appears in the system's package list:

```
$ dpkg --get-selections | grep myapp  
ii myapp                                0.1                                arm64
```

We can also check if it resides in the specified directory:

```
$ which myapp  
/usr/bin/myapp  
$ ls -la /usr/bin/ | grep myapp  
-rwxr-xr-x 1 root root      71072 Feb  3 20:15 myapp
```

What is also pretty important, is to test uninstalling it using `dpkg -r`, the `dpkg` should automatically remove all files mentioned in the package Manifest, thus making sure we do not leave any trash behind:

```
$ dpkg -r myapp  
(Reading database ... 16607 files and directories currently installed.)  
Removing myapp (0.1) ...  
$ myapp  
bash: /usr/bin/myapp: No such file or directory  
$ which myapp  
$ dpkg --get-selections | grep myapp  
$ # As you can see above nothing was found because it's deleted :)
```

## What's Next?

This guide covers the basics of packaging a simple C++ application into a `.deb` file. However, the same principles apply to other programming languages and more complex applications. As you grow more comfortable

with the process, consider integrating packaging into your Continuous Integration/Continuous Deployment (CI/CD) pipelines for automated builds and distributions.

## Conclusion

Packaging a C++ application into a `.deb` file might seem daunting at first, but as we've seen, it's a straightforward process. By following these steps, you can package your application, making it easier to distribute and install on Debian-based systems. Remember, the `.deb` format isn't just for C++ applications; it's versatile enough for software written in any language. With this newfound knowledge, you're well on your way to becoming a proficient software packager in the Linux world!

This article is based on my video version of this guide:

How to create a .deb package for Ubuntu/Debian/Pop\_OS/Linux Mint! Qui...





Linux

Programming

Ubuntu

Software Development

Cpp



**Written by Jan Bronicki**

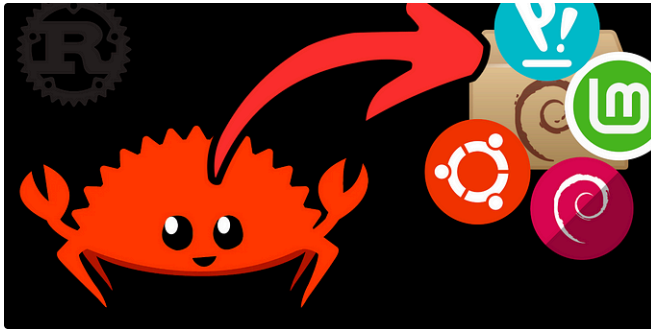
Follow

50 Followers · Editor for CodeCurrent

Tech and open-source enthusiast 🖥️, Engineer 🧑‍💻, Python Developer 🐍, Rustacean 🦀

---

**More from Jan Bronicki and CodeCurrent**



Jan Bronicki in Rust Programming Language

## Simplifying Debian Packaging for Rust: A Step-by-Step Guide for...

In the rapidly evolving world of software development, efficiency and simplicity are...

Feb 3 🖱️ 24



Jan Bronicki in CodeCurrent

## Maximizing Efficiency: The Power of TLP on ThinkPads

Unlocking the Full Potential of Your Linux Laptop

★ Mar 16



Jan Bronicki in CodeCurrent

## Revolutionize Your R Projects: Why renv Might Be the Solution You...

Simplifying R with renv: Your Key to Hassle-Free Data Analysis and Collaboration

★ Feb 14 🖱️ 2 💬 1



Jan Bronicki in Rust Programming Language

## Rust Can Now Automate Vulnerability Detection in Your...

Unlock the Power of Security: Navigating Vulnerabilities in Rust 🦀 🐛

Feb 4 🖱️ 16



See all from Jan Bronicki

See all from CodeCurrent

## Recommended from Medium



Desiree Peralta in Publishous

### OnlyFans is Finally Dead

And I'm happy about it.



Oct 8



15.5K



294



Hayk Simonyan in Level Up Coding

### STOP using Docker Desktop: Faster Alternative Nobody Uses

Ditch Docker Desktop and try this faster, lighter tool that will make your life easier!

Oct 8



1.8K



33



## Lists



### General Coding Knowledge

20 stories · 1663 saves



### Stories to Help You Grow as a Software Developer

19 stories · 1436 saves



### Coding & Development

11 stories · 858 saves



### Leadership

61 stories · 467 saves



 Thuwarakesh Murallie in Towards Data Science

## I Fine-Tuned the Tiny Llama 3.2 1B to Replace GPT-4o

Is the fine-tuning effort worth more than few-shot prompting?

 6d ago  1.7K  19 

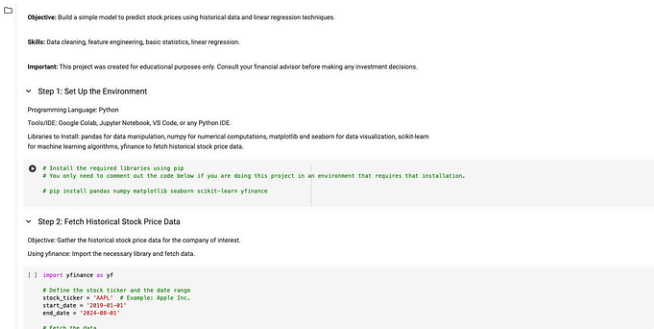



 Kuldeepkumawat

## Build a Beautiful Weather App with Flask and OpenWeather API: A...

Introduction:

 Aug 9  1 

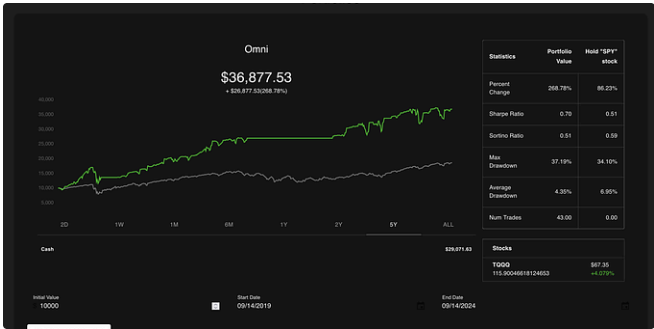



 Leo Mercanti

## Predicting Stock Prices with Machine Learning: An Educationa...

Have you ever been curious about predicting stock prices? With the power of machine...

Aug 30 



 Austin Starks in DataDrivenInvestor

## I used OpenAI's o1 model to develop a trading strategy. It is...

It literally took one try. I was shocked.

 Sep 15  4.3K  118 

See more recommendations