# Sourcecode software installation in Linux

Daniel Calviño Sánchez



## Table of Contents



Brief article explaining the problems and possible solutions of installing free (as in freedom) software from sourcecode in Linux.

# Abstract

Nowadays, Linux distributions have very advanced, useful and complete software packages installation systems. Official packages repositories for the distro are populated with thousands of them, and they're also a lot of unofficial repositories. But sometimes, you need, for some reason, to install a package directly from a sourcecode release.

That, itself, isn't a big problem. Most of free software packages source releases follow the old yet effective "./configure; make; make install" sequence, alhough some use newer build systems as jam. But perhaps you lack some dependencies and have to install them. And you must check for newer source releases to update. Or maybe you want to uninstall them (which can be a pain). Have you updated your distro? Maybe you need to recompile those source installed apps. And what about doing menu entries?

Well, those things aren't very complex to do. But they're tedious. It will be good if they were automatized, at least, to some degree. However, due to the variety of possible options involved in the process, the automatization system must be flexible enough to accomodate them all. A framework for sourcecode installers could be the answer.

# Introduction

Generally, installing software in Linux is incredibly easy. You simply know the name of the package you're interested in... and that's all. The distro packaging system takes care of download and install it and its dependencies. Updating and uninstalling them happens in the same way. The repositories of packages of every distro usually have thousands of them. So, in most cases, you'll have all the needed software in the repositories.

But the key here is "in most cases". You may want a newer version of a package than the one in the repositories (because although there're updates, most of them are only bug and security fixing). Or even a package that isn't at all in the repositories.

When we're talking about main pieces of the system (for example, KDE libraries), it's usually better to wait for an official update than mess the base system with our own updates. But if the package we want isn't a critical part (such as a KDE program as Umbrello) we may want to go on and install that package ourselves.

If we decided to install the package, there are various options. We can look for the package we need in an unofficial repository. Or go to the project's page and see if there is there a package for our distro. Or we can simply grab the source package and compile it. Lets take a look in detail to the compiling option.

# Problem

So we're going to install from source an application. First, we must unpack the source package. Then, install the known dependencies, usually from the official packages repositories for the distro. Run the configure script, if any. If it fails, it'll usually be an unresolved dependency, so we install it. Now the building. Most source packages uses "make", although newer system are begun to be used. Finally, we install the package (usually needing the root password if we want to install it system wide). Finally? Well, not really. You may want to add a menu entry for the app, or even associate some mime types with it. Some of those operations will depend on the distro being used, and even the desktop system.

Well, it was "easy". But it was only installing. What about uninstalling the package? And updating it? Most building system files will have an "uninstall" target (which forces you to keep the building file for the future, because you may need to uninstall the package). But if they haven't it... uninstalling a package will be a pain.

That's why programs such as checkinstall [http://asic-linux.com.mx/~izto/checkinstall/] appeared. A Slackware, RPM or Debian compatible package will be created after the make install with the installed app which will allow to uninstall it using the distribution's packages system. But there are people who prefer to only use the official packages in their distro's packages system to avoid it to be messed up (a reason to install from source code even when there are unofficial packages for the distro).

When it comes to update a package installed from source you must, first of all, know if there was an update, usually visiting the project's page. Then download it and proceed to install. But, first, you must uninstall the previously installed package. And then install the updated one, generally using the same configuration options you used with the previously installed version.

And if you upgrade your distro to a newer version? You'll usually recompile the packages so they use the newer libraries. Perhaps some installed from source package is now available in the distribution's packages system, so you may want to uninstall the installed from source.

All those things aren't very complex (at least, not if you have made them more than once). But are boring, and repetitive. And usually, distribution and desktop system dependent. A friendlier way to manage the source packages installation and related tasks will make Linux users happier (at least, some of them).

# Possible solution

All the mentioned tasks are very straightforward. To accomplish them, you usually follow the same schema. So, why don't get rid of those tasks? It's simply a matter of finding a way to make automatically what is now done manually. An extensible library may fit the desired purpose.

Why a library? Because with a library, the implemented solution can be reused from various apps, you're not tied to a concrete one. So you can have an interface for the library for KDE, or GNOME, or console, or... And there's no need for an specific interface. It could even be integrated with a general packages system.

And, what is the needing of doing it extensible? Couldn't it be done with fixed functionality? The answer is no. For example, installing dependencies using the distribution's packages system is, logically, disitribution dependent. And adding menu entries, desktop dependent (although the freedesktop.org project [http://www.freedesktop.org] is simplifying this). The building tasks depdends on the building system used in the source package. And so on.

But the extensibility doesn't implies only the possibility to create specific implementations of the tasks (unpacking, configuring, building...). The tasks itselfs must be extensible, so new task types

colud be added. For example, a task that downloads a package from the project's webpage. Or a task that creates a binary package for an installed app. Or... well, you can see what I mean. Although initially there seem to be a fixed number of tasks to be done, they can grow so more things than the ones in the basic installation can be done.

The tasks may need being configured. And they may even need parameters specified by the user to do what they're intended for. So although a full automatization won't be possible, at least the amount of manually executed tasks can reduced.

The user specified parameters could be shared among users. So if someone has defined them for a specific combination of variables (package and distro, or desktop system and distro, or...), they can be obtained from internet and used in other user's computer with the same combination of variables. However, this could be added in an advanced stage of development. That is, focus first in accomplish the base functionality, and when this is reached, add this idea.

Although the library can solve the problem of installing from source, there should be a way to use that library. So a default user interface should be made to work with it.

# Conclusion

Installing (and related operations as updating and uninstalling) a package from source isn't very difficult. However, it has some problems, the main of it that it's repetitive and tedious. When you install a package manually, you usually follow the same mental schema. So recreating this schema in a program could help us to get rid of those repetitive tasks.

An extensible library designed for installing, updating and uninstalling packages could be the solution, at least to some degree, of those problems. Human actions will still be necessary in some points, at least until artificial intelligence can be smart enough to install from source without external intervention. But, even still having to help the program to fulfill its mission, it'll simplify a lot the installing from source package process.