

Compiling and installing software from sources

Section 1. Tutorial tips

Should I take this tutorial?

Do I need experience?

If you're relatively new to Linux, or any other Unix or Unix-like operating system, and would like to learn how to compile and install open source programs from their original source code, then this tutorial is for you. In it, you'll learn how to compile the vast majority of Unix sources. The unpacking, inspection, configuration, compilation, and installation steps will be covered in detail.

If you're an intermediate Unix user, and only rarely compile programs from their original sources, you may find this tutorial to be a good refresher course.

Navigation

Navigating through the tutorial is easy:

1. Use the Next and Previous buttons to move forward and backward through the tutorial.
 2. Use the Menu button to return to the tutorial menu.
 3. If you'd like to tell us what you think, use the Feedback button.
 4. If you have a question for the author about the content of the tutorial, use the Contact button.
-

Contact

For technical questions about the content of this tutorial, contact the author, Daniel Robbins, at drobbins@gentoo.org.

Daniel resides in Albuquerque, New Mexico. He is the President/CEO of Gentoo Technologies, Inc., the Chief Architect of the *Gentoo Project* and a contributing author of several books published by MacMillan: *Caldera OpenLinux Unleashed*, *SuSE Linux Unleashed*, and *Samba Unleashed*. Daniel has been involved with computers in some fashion since the second grade, when he was first exposed to the Logo programming language as well as a potentially dangerous dose of Pac Man. This probably explains why he has since served as a Lead Graphic Artist at SONY Electronic Publishing/Psygnosis. Daniel enjoys spending time with his wife, Mary, and his new baby daughter, Hadassah.

Section 2. First steps

Introduction

Let's say that you find a particular application that you'd like to install on your system. Maybe you need to run a very recent version of this program, and this most recent version isn't yet available in a packaging format such as RPM. Perhaps this particular application is only available in source form, or you need to enable certain features of the program that are not enabled in the RPM by default.

Whatever the reason, whether of necessity or simply just because you *want* to compile the program from its sources, this tutorial will show you how.

Downloading

Your first step will be to locate and download the sources that you want to compile. They'll probably be in a single archive with a trailing `.tar.gz`, `tar.Z`, `tar.bz2`, or `.tgz` extension. Go ahead and download the archive with your favorite browser or ftp program. If the program happens to have a Web page, this would be a good time to visit it to familiarize yourself with any installation documentation that may be available.

The program you're installing could depend on the existence of any number of existing programs that may or may not be currently installed on your system. If you know for sure that your program depends on other programs or libraries that are not currently installed, you'll need to get these packages installed first (either from a binary package like RPM or by compiling them from their sources too). Then, you'll be in a great position to get your original source file successfully installed.

Unpacking

Unpacking the source archive is relatively easy. If the name of your archive ends with `.tar.gz`, `.tar.Z`, or `.tgz`, you should be able to unpack the archive by typing:

```
$ tar xzvf archivename.tar.gz
```

(`x` is for extract, `z` is for gzip decompression, `v` is for verbose (print the files that are extracted), and `f` means that the filename will appear next on the command line.)

Nearly all "source tarballs" will create one main directory that contains all the program's sources. This way, when you unpack the archive, your current working directory isn't cluttered with lots of files -- instead, all files are neatly organized and don't get in the way.

Listing archives

Every now and then, you may come across an archive that, when decompressed, creates tons of files in your current working directory. While most tarballs aren't created this way, it's been known to happen. If you want to verify that your particular tarball was put together correctly and creates a main directory to hold the sources, you can view its contents by typing:

```
$ tar tzvf archivename.tar.gz | more
(t is for a text listing of the archive. No extraction occurs.)
```

If there is no common directory listed on the left side of the archive listing, you'll want to create a new directory, move the tarball inside it, enter the directory, and only then extract the tarball. Otherwise, you'll be in for a mess!

Unpacking bzip2-compressed archives

It's possible that your archive may be in .tar.bz2 format. Archives with this extension have been compressed with bzip2. Bzip2 generally compresses significantly better than gzip -- its only disadvantage is that compression and decompression are slower, and bzip2 consumes more memory than gzip while running. For modern computers, this isn't much of an issue, which is why you can expect bzip2 to become more and more popular as time goes on.

Because bzip2 has been gaining popularity, many Linux distributions come with versions of tar that have been patched so that passing "y" or "i" option will inform tar that the archive is in bzip2 format and needs to be automatically decompressed with the "bzip2" program. To see if you have a patched version of tar, try typing:

```
$ tar tyvf archive.tar.bz2 | more
or
```

```
$ tar tivf archive.tar.bz2 | more
```

If neither of these commands work (and tar complains of an invalid argument), there is still hope -- read on.

bzip2 pipelines

So, your version of tar doesn't recognize those handy bzip2 shortcuts -- what can be done? Fortunately, there's an easy way to extract the contents of bzip2 tarballs that will work on nearly all Unix systems, even if the system in question happens to have a non-GNU version of tar. To extract a bzip2 file, we can create a pipeline:

```
$ cat archive.tar.bz2 | bzip2 -d | tar tvf - | most
```

The above pipeline will display the contents of archive.tar.bz2.

```
$ cat archive.tar.bz2 | bzip2 -d | tar xvf -
```

The above pipeline will extract the contents of archive.tar.bz2.

bzip2 pipelines (continued)

In the previous two examples, we created a standard Unix pipeline. The vertical bar character tells the shell to "pipe" the output of the command on the left side of the bar to the input of the command on the right side of the bar.

The "cat" command dumps the contents of archive.tar.bz2 to standard output. This data is piped to bzip2's stdin. Then, bzip2 reads the data in, and dumps the decompressed data to stdout. bzip2's stdout is redirected to tar. Since tar is called with the "f -" option, it knows to read tar data from stdin, rather than from file on disk. Then, tar either displays or extracts the contents of the archive, depending on how it was called.

No bzip2!

If you used the pipeline method to try to extract the contents of your archive and your system complained that bzip2 couldn't be found, it's possible that bzip2 isn't installed on your system. You can download the sources to bzip2 from <http://sourceware.cygnum.com/bzip2>. After installing the bzip2 sources (by following this tutorial), you'll then be able to unpack and install the application you wanted to install in the first place.

Section 3. Inspection and configuration

Inspecting sources

Once you've unpacked your sources, you'll want to enter the unpacked directory and check things out. It's always a good idea to locate any installation-related documentation. Typically, this information can be found in a README or INSTALL file located in the main source directory. Additionally, look for README.platform and INSTALL.platform files, where platform is the name of your particular operating system.

Configuration

Many modern sources contain a "configure" script in the main source directory. This script is specially designed to set up the sources so that they compile perfectly on your system. When run, the configure script probes your system, determining its capabilities, and creates "makefiles" that contain instructions for building and installing the sources on your system.

The configure script is always called "configure". If you find one in the main source directory, you'll want to use it to configure your sources for compilation. If you can't find a "configure" script, then your sources probably come with a standard "Makefile" that has been designed to work across a variety of systems -- this means that you can skip the following configuration steps, and resume this tutorial where we start talking about "make".

Using configure

Before running the configure script, it's a good idea to get familiar with it. By typing `./configure --help`, you can view all the various configuration options that are available for your program. Many of the options, especially the ones listed at the top, are standard options that will be found in nearly every configure script. The options listed near the end are often related to the particular package you're trying to compile. Take a look at them and note any ones you'd like to enable or disable.

The --prefix option

Most GNU autoconf-based configure scripts have a "--prefix" option that allows you to control where your program gets installed. By default, most sources install into the `/usr/local` prefix -- this means that binaries end up in `/usr/local/bin`, man pages in `/usr/local/man`, etc. This is normally what you want -- `/usr/local` is commonly used to store programs that you compile by hand.

Using --prefix

If you'd like the sources to install somewhere else, say in /usr, then you'll want to pass the "--prefix=/usr" option to configure. Likewise, you could also tell configure to install to your /opt tree, by using the "--prefix=/opt" option.

Time to configure

Once you've taken a look at the various configure options and determined which ones you'd like to use, it's time to run configure. Please note that it's possible that you won't need to include any command-line options when you run configure -- in the majority of situations, the defaults will work (but may not be *exactly* what you want).

Time to configure (continued)

To run configure, type:

```
$ ./configure <options>
```

This might look like:

```
$ ./configure
```

or

```
$ ./configure --prefix=/usr --with-threads
```

It all depends on the particular package you're configuring. When you run configure, it will spend a minute or two detecting what particular features or tools are available on your system, printing out the results of its various configuration checks as it goes.

config.cache

Once the configuration process is complete, the configure script stores all its configuration data in a file called "config.cache". This file lives in the same directory as the configure script itself. If you ever need to run "./configure" again after you've updated your system configuration, make sure you "rm config.cache" first; otherwise, configure will simply use the old settings without rechecking your system.

Progress quiz

Let's see how well you were paying attention :) True or false: when you call `./configure`, you must always specify the `--prefix` option.

- A. True
- B. False

(The correct answer is "B. False")

Section 4. Compilation and installation

configure and Makefiles

After the configuration completes, it's time to compile the sources into a running program. A program called "make" is used to perform this step. If your software package contained a "configure" script, then when you ran it, "configure" created things called "Makefiles" that were specially customized for your system. These files will tell the make program how to build the sources and install the results.

Makefile intro

Makefiles are typically called "makefile" or "Makefile". There will normally be one makefile in each directory that contains source files, including the main program directory. Make works by figuring out which parts of a program need to be compiled, and in what order. The software developer creates Makefiles that contain instructions (officially called *rules*) to build certain *targets*, like the program you want to install.

Invoking make

Invoking make is easy, just type "make" in the current directory. The make program will automatically look for and read in a makefile called "makefile" or "Makefile" in the current directory. If you type "make" all by itself, it will build the *default target*. Developers normally set up their makefiles so that the default target compiles all the sources:

```
$ make
```

Some makefiles won't have a default target, and you'll need to type this compilation started:

```
$ make all
```

After typing one of these commands, your computer will spend several minutes compiling your program into object code. Assuming it completes with no errors, you'll be ready to install the compiled program on to your system.

Installation

After the program is compiled, there's one more important step -- installation. Although the program is compiled, it's not yet ready for you to use -- all its components need to be copied to the correct locations on your filesystem. For example, all binaries need to be copied to `/usr/local/bin`, and all man pages need to be installed into `/usr/local/man`, etc.

Before you can install the software, you'll need to become root. This is typically done by either logging in as root on a separate terminal, or typing "su", at which point you'll be prompted for root's password. After typing it in, you'll have root privileges until you exit from your current shell session by typing "exit" or hitting control-D. If you're already root, you're ready to go!

make install

Installing sources is easy. In the main source directory, simply type:

```
# make install
```

Typing "make install" will cause the installation process to begin -- files and directories will be copied to your `/usr/local` tree. Depending on the size of the program, this may take anywhere from several seconds to a few minutes.

In addition, "make install" will make sure the installed files have the correct ownership and permissions. After "make install" completes successfully, the program is installed!

Section 5. Post-installation and troubleshooting

Once it's installed

Now that your program is installed, what's next? Running it, of course! If you're not too familiar with how to use the program you just installed, you'll want to read the program's man page by typing:

```
$ man programname
```

Your program may require additional configuration steps, depending on what type it is. For example, if you installed a Web server, you'll need to configure it to start automatically when your system boots. You may also need to set up a configuration file for your Web server -- to configure CGI support, for example.

More info

Some programs don't include man pages, but include info documentation instead. These programs normally include a basic "stub" man page, referring users to the info documentation. To see if your program installed any info documentation, type the following command:

```
$ info programname
```

If documentation was available, it'll be displayed. If no documentation was available, "info" may have found the closest match to what you were looking for. This may not be helpful to you, so make sure to look at the title of the documentation before you start reading it!

Ta da!

Now that you've fully installed a particular software package from its sources, you can now run it. To start the program, type:

```
$ programname  
Congratulations!
```

Possible problems

It's very possible that "configure" or "make", or possibly even "make install" aborted with some kind of error code. This section is intended to help you correct several common problems.

Missing libraries

Every now and then, you may experience a problem where "configure" bombs out because you don't have a certain library installed. To continue the build process, you'll need to temporarily put your current program configuration on hold and track down the sources to the library.

If you're using a packaging system such as RPM, you may want to check to see whether an RPM of the needed library is available. Whether you install the library from a binary package or from sources is personal preference -- either should work. Once the library is installed, make sure to type "ldconfig" as root (under Linux) so that the system can detect the newly installed library.

Then, it's time to get back to your original program. If it bombed out during "configure", run configure again. If it bombed out during a "make", type "make" again.

Other problems

Sometimes, you'll run into some kind of error that you simply don't know how to fix. As your experience with Unix/Linux grows, you'll be able to diagnose more and more seemingly cryptic error conditions.

Sometimes, errors happen because your installed compiler is too old (or even possibly too new!) Other times, the problem you're having is actually the fault of the developers, who may not have anticipated their program running on a system such as yours -- or maybe they just made a typo.

Other problems (continued)

For problems such as these, use your best judgment to determine where to go for help. If this is your first attempt at compiling a program from source, this may be a good time to select another, "easier" program to compile. Once you get the simpler program compiled, you may have the necessary experience to fix your original problem. As you continue to learn more about how Unix works, you'll get closer to the point where you can actually "tweak" Makefiles and sources to get even flaky code to compile!

Section 6. Wrapup

Your feedback

Please let us know whether this tutorial was helpful to you and how we could make it better. We'd also like to hear about other tutorial topics you'd like to see covered.

Thanks!