Installing Open-Source Software on Linux

GOING TO THE SOURCE

CHRIS BROWN

Sometimes we need to get new programs for the Linux system, but installing them can be a chore. This month we look at installation from the source.

Installing from source code

The idea of installing from source code might seem daunting – all those nasty curly brackets and stuff. Isn't this option out of the question for the non-programmers amongst us? No, certainly not! Installing from source usually does not require you to modify, understand, or even look at the actual source code at all. However, since most of the software is written in C or C++, it does require that you install the C/C++ development tools on your system.

Most of the established open-source sites use a format known as a tarball, (also known as a compressed tar archive) to package source code for distribution. These files usually have names ending in .tar.gz. As an example, we're going to take a look at installing the latest version of the Apache Web server from source. The process is much the same for other packages.

As I'm writing this article, the most recent version of Apache I can find as an RPM is 1.3.20; however I know that there's a beta version of 2.0 available. We'll try the obvious place – *www.apache.org*. Sure enough, after a couple of minutes poking around I find a listing of files available for download (see Figure 1). The tarball I need is called *httpd-2_0_16-beta.tar.gz*. The file below it in the list is a PGP signature for the file, so I can be sure it's authentic. I decide to download the tarball to my home directory, */home/chris*.

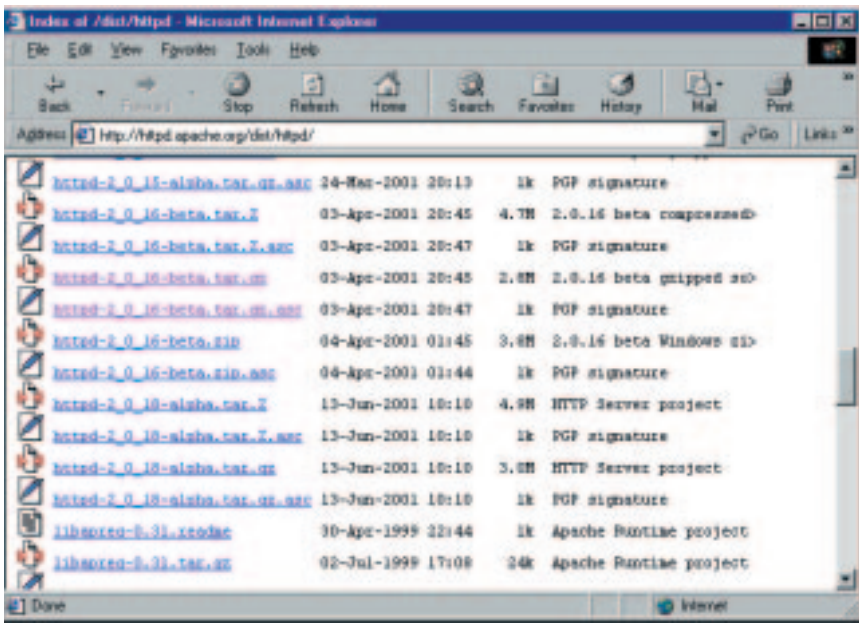

Figure 1: Finding files

The next thing to do is to uncompress and unpack the archive. With the right switches, tar can do both of these in one step:

```
$ cd /home/chris
$ tar xzvf httpd-2_0_16-beta.tar.gz
```

You'll see a long list of the files as tar extracts them from the archive and puts them into the subdirectory *httpd-2_0_16*. If you'll look in that directory you'll see some documentation files with names like INSTALL and README, which you'll probably want to look at.

Now it's time to build the software. Not many years ago this typically involved quite a bit of fiddling around to customise the build process to your platform, following instructions in the INSTALL file which said lots of intimidating things like "if you don't have the library libfoobar.o, add the flag -DNOFOOBAR to the CFLAGS macro definition in the makefile". Nowadays this customisation is mostly automatic thanks to an amazing tool called autoconf from the Free Software Foundation. Autoconf is used by the package developer to create a script called configure which is included in the tarball. The configure script performs lots of tests on your system and builds a makefile depending on what it finds. By the way, I do not recommend actually looking at the configure script; like most automatically generated code, it's not a pretty sight.

If configure finds that necessary components are absent from your system, it will tell you what's missing and abort. In any event, you'll see a long list of all the tests that the configure script is making scroll by. If all goes well, you end up with a makefile to control the build of the software.

There's not space enough here to talk about makefiles and the make command in depth. Suffice for now to say that the makefile specifies what files need to be created, which files they need to be created from, and what commands are needed to do the job. The make program interprets the makefile and runs the necessary commands. Usually all you need to do is to run make with no arguments. This will compile and link the programs that make up the package, and may take some time depending on the complexity of the package and the speed of your computer. Time to go and top up the bird feeders with peanuts perhaps. If configure ran successfully, the make is unlikely to fail.

Now you have the compiled version of the package. Note that everything so far has been contained within the directory you did the build in – in our example, that's */home/chris/httpd-2_0_16*. If I were to empty and remove this directory, and delete the tarball from my home directory, I would remove all traces of the package.

The final step is to install all the pieces into the correct places in your system. This might be as simple as putting the program into */usr/bin* for example, but will typically also install documentation and maybe some configuration files. Because this operation updates system directories, it must be run as root. This operation is also automated via entries in the makefile and the command is simply 'make install'.

In most cases, that's all you'll need to do. It takes longer to explain than to actually do. In summary, the sequence of commands is usually:

```
<download the tarball to /somewhere >
$ cd /somewhere
$ tar xzvf package_name.tar.gz
$ cd package_name
$ ./configure
$ make
  su to root ...
# make install
```

Once the package is installed you can recover some disk space by deleting the directory you unpacked the tarball into – for example:

```
$ rm -rf /somewhere/package_name
```

Installing new software onto Linux isn't hard and doesn't require any programming skills. It's an excellent way of expanding your system and keeping what you have up to date. And of course, it's free!

Happy hunting! ■