Compiling-Software Tips

Aaron Ucko, ucko@vax1.rockhurst.edu v1.1, March 14, 1996

1 Preface

"Red Hat Tips" are documentation meant to help Red Hat users with specific tasks. Some of these documents are for new users, some are for advanced users. Hopefully each document will also be of help for both new and advanced users. If you have contributions to make, please send them to tech-sup@redhat.com. If you have changes that need to be made to individual Tips, send them to the author of that document.

2 Introduction

This document explains how to compile software written in C, C++, or Objective C. It also deals with related issues such as applying patches and creating new RPMs.

3 Packages to Install

The following packages are absolutely required to compile anything:

- `gcc`: the compiler
- `binutils`: supporting utilities
- `kernel-source`: it contains some important headers
- `libc-include`: it contains most other headers
- `libc-devel`: the remaining miscellanea

In addition, you'll almost certainly want to have `make`, which allows you to use `Makefiles`.

Other possibly necessary packages include

- `bin86`: real-mode binutils (used to build the kernel and `dosemu`)
- `bison`: the GNU version of `yacc`, a parser generator (used for `*.y`)
- `byacc`: the BSD version of `yacc` (a few programs prefer this to `bison`)
- `flex`: the GNU version of `lex`, a lexical scanner generator (used for `*.l`)

- gcc-c++: the C++ compiler (used for *.cc, *.C, *.cxx, and *.c++)
- libg++: the standard and GNU c++ libraries (needed for almost all C++ programs)
- gcc-objc: the Objective-C compiler (used for *.m)
- patch: a patch applier
- diffutils: a patch maker
- pmake: a BSD version of make (BSD Makefiles have a slightly different syntax)
- XFree86-devel: XFree86 libraries (needed to build X programs)
- ImageMagick-devel: ImageMagick libraries (rarely needed)
- e2fsprogs-devel: ext2fs libraries (needed to build low-level filesystem programs)
- libgr-devel: various graphics libraries (needed by many graphics programs)
- ncurses-devel: a full-featured curses library (needed by many interactive textual programs)
- slang-devel: the SLang library (used by some programs)
- svgalib-devel: a console graphics library (used by quite a few programs)

4 Compiling the kernel

Contrary to popular belief, this is not all that difficult. Once you have patched (see the section on patching) and/or edited it to suit your system, just configure and build it. For more details, see the Kernel HOWTO, which is available in usr/doc/HOWTO/Kernel-HOWTO.gz if you have installed the package howto. (You can view it without decompressing it by using zmore. If the package is not installed and your Red Hat CDROM is mounted on /mnt/cdrom, type "rpm -ivh /mnt/cdrom/RedHat/RPMS/howto-[1-9]*.rpm")

A few notes: In recent kernels, you can type "make xconfig" or "make menuconfig" to run X11 and curses configuration programs respectively. Both of these provide optional help for their questions, as do newer versions of the regular configuration script. Other documentation, which may be newer and/or more complete than that in HOWTOs, can be found in /usr/src/linux/Documentation.

5 Compiling other software

Always check files called README, INSTALL, or something equally prominent for program-specific instructions. In general, however, the presence of an Imakefile indicates that you should probably run

```
xmkmf -a
make all
```

and the presence of a Makefile.in indicates that you should try

```
 ./configure --prefix=/usr
make all
```

If there is just a Makefile, try executing "make all", "make", and "make *progname*" in that order. Otherwise, you'll have to invoke gcc manually: if the source is in foo.c, try

```
gcc -c foo.c -o foo.o
gcc foo.o -o foo
```

to produce an executable called foo.

6 Applying patches

If the patch is compressed, decompress it. Don't worry about stripping any leading text (e-mail headers, etc.); patch will do that itself. The patch itself should start with a path name and a timestamp and then another line with a different path name and timestamp. See how many directories you need to drop from the second path name in order to get a name relative to the source directory of the program you want to patch. (See patch's manpage or info file for examples and a more detailed explanation.) Now change to the program's source directory and type patch -s -p*n* < *patchfile*, where *n* is the number of directories you need to drop and *patchfile* is the path to the patch you want to supply. Do not omit the -p flag even if *n* is zero; otherwise, directory structure will not be preserved correctly.

If patch prompts you for a file to patch, you probably got *n* wrong; take another look at the patch and try again (leading slashes count too). If it complains about hunks failing, do not panic. Look at the resulting .rej files one by one, figure out what changes they are trying to make, and make them yourself. If you feel uncomfortable doing this, ask somebody more experienced or politely post a query to an appropriate mailing list or newsgroup.

7 Creating RPMs

Let us suppose that you have a program which you would like to turn in to an RPM. You might need to make some minor changes to the program to customize it for Red Hat Linux. Once you have made any necessary changes, you will need to generate one or more patches for RPM to use.

You can use gendiff to make the patches for your RPM. Just make sure that for every file foo you modify, you keep an original version of that file under a name like foo.dist. If you have done this, and your sources are in the directory newpkg-1.7, just type "gendiff newpkg-1.7 .dist > mypatch" and you'll get a patch you can use (with -p1 in this case) in your RPM.

If you have made two independent changes (unrelated changes with the additional property that neither affects a file affected by the other) you can rename original files foo affected by the first change foo.dist-1 and files bar affected by the second bar.dist-2; then you can type "gendiff newpkg-1.7 .dist-1 > mypatch-1" and "gendiff newpkg-1.7 .dist-2 > mypatch-2" to obtain two separate patches with very little work.

Then just create a specification file according to the directions in the RPM HOWTO, which gives numerous examples at varying levels. Note that the `Icon` field does not really need to be filled in; if it is left blank, `glint` will use a default "package" icon.

8 Copyright Notice