Using Cscope on large projects (example: the Linux kernel)

Cscope can be a particularly useful tool if you need to wade into a large code base. You can save yourself a lot of time by being able to do fast, targeted searches rather than randomly grepping through the source files by hand (especially since grep starts to take a while with a truly large code base).

In this tutorial you'll learn how to set up Cscope with a large project. We'll use as our example the Linux kernel source code, but the basic steps are the same for any other large project, including C++ or Java projects.

1. **Get the source**. First get the source code. You can download the Linux kernel source from http://www.kernel.org. For the rest of this tutorial, I'll assume you've downloaded Linux 2.4.18 and installed it into `/home/jru/linux-2.4.18`.

   Note: Make sure you've got enough disk space: the kernel tarball alone is 30 MB, it expands into 150 MB of source code, and the Cscope database we'll generate will gobble up another 20-100+ MB (depending on how much of the kernel code you decide to include in the database). You can put the Cscope database on a different disk partition than the source code if you need to.

2. **Figure out where you want to put your Cscope database files**. I'll assume you'll use `/home/jru/cscope` as the directory to store your database and associated files.

3. **Generate `cscope.files` with a list of files to be scanned**. For some projects, you may want to include every C source file in the project's directories in your Cscope database. In that case you can skip this step, and just use '`cscope -R`' in the project's top-level directory to build your Cscope database. But if there's some code that you wish to exclude, and/or your project contains C++ or Java source code (by default Cscope only parses files with the `.c`, `.h`, `.y`, or `.l` extensions), you'll need to generate a file called `cscope.files`, which should contain the name of all files that you wish to have Cscope scan (one file name per line).

   You'll probably want to use absolute paths (at least if you're planning to use the Cscope database within an editor), so that you can use the database from directories other than the one you create. The commands I show will first cd to root, so that find prints out absolute paths.

   For many projects, your find command may be as as simple as

   ```
   cd /
   find /my/project/dir -name '*.java' >/my/cscope/dir/cscope.files
   ```

   For the Linux kernel, it's a little trickier, since we want to exclude all the code in the docs and scripts directories, plus all of the architecture and assembly code for all chips except for the beloved Intel x86 (which I'm guessing is the architecture you're interested in). Additionally, I'm excluding all kernel driver code in this example (they more than double the amount of code to be parsed, which bloats the Cscope database, and they contain many duplicate definitions, which often makes searching harder. If you are interested in the driver code, omit the relevant line below, or modify it to print out only the driver files you're interested in):

   ```
   LNX=/home/jru/linux-2.4.18
   cd /
   find  $LNX                                                              \
       -path "$LNX/arch/*" ! -path "$LNX/arch/i386*" -prune -o             \
       -path "$LNX/include/asm-*" ! -path "$LNX/include/asm-i386*" -prune -o \
       -path "$LNX/tmp*" -prune -o                                         \
       -path "$LNX/Documentation*" -prune -o                               \
       -path "$LNX/scripts*" -prune -o                                     \
       -path "$LNX/drivers*" -prune -o                                     \
   ```

```
-name "*.[chxsS]" -print >/home/jru/cscope/cscope.files
```

While find commands can be a little tricky to write, for large projects they are much easier than editing a list of files manually, and you can also cut and paste a solution from someone else.

4. **Generate the Cscope database**. Now it's time to generate the Cscope database:

```
cd /home/jru/cscope     # the directory with 'cscope.files'
cscope -b -q -k
```

The `-b` flag tells Cscope to just build the database, and not launch the Cscope GUI. The `-q` causes an additional, 'inverted index' file to be created, which makes searches run much faster for large databases. Finally, `-k` sets Cscope's 'kernel' mode--it will not look in `/usr/include` for any header files that are #included in your source files (this is mainly useful when you are using Cscope with operating system and/or C library source code, as we are here).

On my 900 MHz Pentium III system (with a standard IDE disk), parsing this subset of the Linux source takes only 12 seconds, and results in 3 files (`cscope.out`, `cscope.in.out`, and `cscope.po.out`) that take up a total of 25 megabytes.

5. **Using the database**. If you like to use vim or emacs/xemacs, I recommend that you learn how to run Cscope within one of these editors, which will allow you to run searches easily within your editor. We have a tutorial for Vim, and emacs users will of course be clever enough to figure everything out from the helpful comments in the `cscope/contrib/xcscope/` directory of the Cscope distribution.

Otherwise, you can use the standalone Cscope curses-based GUI, which lets you run searches, then launch your favorite editor (i.e., whatever $EDITOR is set to in your environment, or 'vi' by default) to open on the exact line of the search result.

If you use the standalone Cscope browser, make sure to invoke it via

```
cscope -d
```

This tells Cscope not to regenerate the database. Otherwise you'll have to wait while Cscope checks for modified files, which can take a while for large projects, even when no files have changed. If you accidentally run '`cscope`', without any flags, you will also cause the database to be recreated from scratch without the fast index or kernel modes being used, so you'll probably need to rerun your original cscope command above to correctly recreate the database.

6. **Regenerating the database when the source code changes**.

If there are new files in your project, rerun your 'find' command to update `cscope.files` if you're using it.

Then simply invoke `cscope` the same way (and in the same directory) as you did to generate the database initially (i.e., `cscope -b -q -k`).

Tutorial by Jason Duell

Back to the Cscope home page