The Vim/Cscope tutorial

Cscope is a very handy tool, but it's even better when you don't ever have to leave the comfort of your favorite editor (i.e. Vim) to use it. Fortunately, Cscope support has been built into Vim.

This tutorial introduces you both to Vim's built-in Cscope support, and to a set of maps that make searching more convenient.

It is assumed you know the basics of using a vi-style editor, but you don't need any particular knowledge about Vim (where Vim-specific features--like multiple windows--are used, a working knowledge of the features is briefly introduced). You also don't need to know anything about Cscope: the basics are introduced as we go along.

In a nutshell, Vim's Cscope support is very similar to Vim's ctags features, in case you've used those. But since Cscope has more search types than ctags, there are a few differences.

This is a hands-on tutorial, so open up a shell, and follow these steps:

1. Get and install Cscope if you don't have it already on your machine. Ideally, you will also have Vim 6.x, but you can get most of the functionality with later versions of Vim 5 (vertical splits don't work, but horizontal splits will work if you modify the maps as described in the file's comments).

   Note: If your version of Vim wasn't compiled with '--enable-cscope', you will need to reconfigure and recompile Vim with that flag. Most Vim binaries that ship with Linux distributions have the Cscope plugin enabled.

2. Download the cscope_maps.vim file, and arrange for it to be read by Vim at startup time. If you are using Vim 6.x, stick the file in your $HOME/.vim/plugin directory (or in any other 'plugin' subdirectory in your 'runtimepath'). If you are using Vim 5.x, you can either cut and paste the entire contents of the cscope_maps file into your $HOME/.vimrc file, or stick a "source cscope_maps.vim" line into your .vimrc file.

3. Go into a directory with some C code in it, and enter 'cscope -R' (the '-R' makes Cscope parse all subdirectories, not just the current directory). Since we aren't passing the '-b' flag (which tells Cscope to just build the database, then exit), you will also find yourself inside Cscope's curses-based GUI. Try a couple of searches (hint: you use the arrow keys to move around between search types, and 'tab' to switch between the search types and your search results). Hit the number at the far left of a search result, and Cscope will open Vim right to that location. (unless you've set your EDITOR environment variable to something besides Vim). Exit Vim, and you'll be right back in the Cscope GUI where you left off. Nifty.

   Alas, the Cscope interface has one big problem: you need to exit Vim each time you want to do a new search. That's where the Vim plugin comes in. Hit CTRL-D to exit Cscope.

4. Start up Vim. If you want, you can start it with a C symbol (ex: 'vim -t main'), and you should hop right to the definition of that symbol in your code.

5. Put the cursor over a C symbol that is used in several places in your program. Type "CTRL-\ s" (Control-backslash, then just 's') in quick succession, and you should see a menu at the bottom of your Vim window showing you all the uses of the symbol in the program. Select one of them and hit enter, and you'll jump to that use. As with ctags, you can hit "CTRL-t" to jump back to your original location before the search (and you can nest searches and CTRL-t will unwind them one at a time).

   Mnemonic: the '\' key is right next to the ']' key, which is used for ctags searches.

6. Try the same search, but this time via "CTRL-spacebar s". This time, your Vim window will split in two horizontally , and the Cscope search result will be put in the new window. [if you've never used multiple Vim windows before: move between windows via 'CTRL-W w' (or CTRL-W arrow key, or CTRL-W h/j/k/l for left/up/down/right), close a window via 'CTRL-W c' (or good old ':q'), make the current window the only one via 'CTRL-W o', split a window into two via 'CTRL-W s' (or 'CTRL-W v' for a vertical split), open a file in a new window via ':spl[it] filename']

   Mnemonic: there's now a big, spacebar-like bar across the middle of your screen separating your Vim windows.

7. Now try the same search via "CTRL-spacebar CTRL-spacebar s" (just hold down the CTRL key and tap the spacebar twice). If you have trouble hitting the keys fast enough for this to work, go into the cscope_maps.vim script and change Vim's timeout settings as described in the comments [actually, I generally recommend that you turn off Vim's timeouts]. This time your Vim window will be split vertically (note: this doesn't work with Vim 5.x, as vertical splits are new with Vim 6.0).

8. Up to now we've only been using the keystroke maps from 'cscope_maps.vim', which all do a search for the term that happens to be under your cursor in Vim. To do Cscope searches the old-fashioned way (using Vim's built-in Cscope support), enter ":cscope find symbol foo" (or, more tersely, ":cs f s foo"). To do the horizontal split version, use ":scscope" (or just ":scs") instead (Vim 6.x only). While it's easier to use the maps if the word you want to search for is under your cursor, the command line interface lets you go to any symbol you type in, so you'll definitely want to use it at times.

9. So far we've only been doing one kind of search: 's', for 'find all uses of symbol X'. Try doing one of Cscope's other searches by using a different letter: 'g' finds the global definition(s) of a symbol, 'c' finds all calls to a function, 'f' opens the filename under the cursor (note: since Cscope by default parses all C header files it finds in /usr/include, you can open up most standard include files with this). Those are the ones I use most frequently, but there are others (look in the cscope_maps.vim file for all of them, and/or read the Cscope man page).

10. Although Cscope was originally intended only for use with C code, it's actually a very flexible tool that works well with languages like C++ and Java. You can think of it as a generic 'grep' database, with the ability to recognize certain additional constructs like function calls and variable definitions. By default Cscope only parses C, lex, and yacc files (.c, .h, .l, .y) in the current directory (and subdirectories, if you pass the -R flag), and there's currently no way to change that list of file extensions (yes, we ought to change that). So instead you have to make a list of the files that you want to parse, and call it 'cscope.files' (you can call it anything you want if you invoke 'cscope -i foofile'). An easy (and very flexible) way to do this is via the trusty Unix 'find' command:

    ```
    find . -name '*.java' > cscope.files
    ```

    Now run 'cscope -b' to rebuild the database (the -b just builds the database without launching the Cscope GUI), and you'll be able to browse all the symbols in your Java files. Apparently there are

folks out there using Cscope to browse and edit large volumes of documentation files, which shows how flexible Cscope's parser is.

For larger projects, you may additionally need to use the `-q` flag, and/or use a more sophisticated 'find' command. See our tutorial on using Cscope with large projects for more info.

11. Try setting the $CSCOPE_DB environment variable to point to a Cscope database you create, so you won't always need to launch Vim in the same directory as the database. This is particularly useful for projects where code is split into multiple subdirectories. Note: for this to work, you should build the database with absolute pathnames: cd to /, and do

    ```
    find /my/project/dir -name '*.c' -o -name '*.h' > /foo/cscope.files
    ```

    Then run Cscope in the same directory as the cscope.files file (or use 'cscope -i /foo/cscope.files'), then set and export the $CSCOPE_DB variable, pointing it to the cscope.out file that results):

    ```
    cd /foo
    cscope -b
    CSCOPE_DB=/foo/cscope.out; export CSCOPE_DB
    ```

    (The last command above is for Bourne/Korn/Bash shells: I've forgotten how to export variables in csh-based shells, since I avoid them like the plague).

    You should now be able to run 'vim -t foo' in any directory on your machine and have Vim jump right to the definition of 'foo'. I tend to write little shell scripts (that just define and export CSCOPE_DB) for all my different projects, which lets me switch between them with a simple 'source projectA' command.

BUG: in versions of Cscope prior to 15.4, there is a silly bug that may cause Vim to freeze when you do this unless you call your database something other than the default 'cscope.out': use '-f foo' in your Cscope invocation to name your database 'foo.out' instead, and you'll be OK.

12. That's it! Use ":help cscope" (in Vim) and/or "man cscope" (from your shell) if you've got questions, and to learn the fine points.

Cscope support added to Vim by Andy Kahn
Tutorial and cscope_maps.vim by Jason Duell
Cscope art by Petr Sorfa

Back to the Cscope home page