Using ctags to Navigate Source Code

A common request of developers who are new to Linux is for a utility which enables them to rapidly locate function names, class definitions, macro definitions and so on in a large piece of code. While people sometimes struggle with tools like grep as a solution, there is actually a special utility designed to solve this exact problem. It's called "ctags", and though it has been around for many years, even many seasoned Linux developers are unaware of its abilities and usefulness.

ctags works by looking at all the source files for a piece of code and finding the lines which contain symbol definitions – that is, it locates the places where the functions, variables and so on are defined. It has two distinct advantages over a more general approach using something like grep: firstly ctags understands the syntax of many languages, which allows it to find the symbols very reliably. Secondly, its output is a file in a format which can be used by many of today's more powerful text editors to jump straight to required symbol definitions. Even in a huge piece of code spread out over hundreds of source files, finding a function is never more than a couple of key strokes away.

This article will show what ctags is, how to use it, and how to make use of its output in a text editor.

The Versions of ctags

The idea of ctags has been around for quite a while, and in the beginning there were actually two versions in common use. "ctags" was the name given to the utility which was used with the vi editor, while "etags" was created for use with the Emacs editor. Although the purpose, usage and approach of the two utilities were just about identical, there were just enough differences between them to ensure that a single solution couldn't materialise.

Fortunately things didn't get any more complicated, and have now been mostly sorted out. This is largely due to a de-facto standard ctags utility emerging in the shape of "Exuberant Ctags". Exuberant Ctags works happily in either ctags mode (for vi and related editors) and in etags mode (for Emacs and related editors). Its functionality and feature set put it way above the original utilities, and virtually every modern Linux distribution supplies it as the standard ctags solution.

Many Linux distributions also make the original etags utility available, normally as part of the Emacs editor package. On the latest Redhat, SUSE and Mandrake distributions, running "ctags --version" shows that Exuberant Ctags is installed as the default ctags package. The default etags utility, however, is often the one which comes with the Emacs editor. Since Exuberant Ctags in Emacs mode is widely considered to be superior to the etags program which comes with Emacs, this article assumes that the user is running Exuberant Ctags regardless of which editor they use.

It should be noted that the command line flags for the Exuberant Ctags ctags command running in Emacs mode differ from those used by the Emacs etags utility. The differences would only be noticed if running, for example, a third party script which assumes the Emacs version of etags. For the majority of users Exuberant Ctags is the only version of ctags or etags ever needed.

So what does ctags actually do?

ctags builds what is known as a “tags file” (also sometimes called a “tags table”). This is a plain text file which contains one line per symbol in the source code. Each line contains the symbol, the file and line number where it is defined, and some extension fields, the contents of which depend on the language and nature of the symbol. The information from this tags file can be loaded into a suitably capable text editor. When the user wants to find the definition of a given function, class or other symbol, they just hit the right button, type in the name of the required symbol, and the text editor can jump straight to it, opening the file if necessary.

Creating a tags file involves a parse over the entire source code tree. Each source file will be opened, its language ascertained, then examined for symbols of interest. It follows, therefore, that ctags must have some fairly complicated parsing logic inside it, and that is indeed the case. In fact, it has lots, since the latest version of Exuberant Ctags (5.5.2 as of this writing) can parse and extract the relevant symbols from no fewer than 33 different programming languages.

A tags file is a static table, and therefore requires rebuilding as the source code changes. Exactly how frequently this needs to happen depends on the piece of code and the nature of the changes which are happening to it. Fortunately ctags parses code and builds tags files very quickly, so a tags file rebuild is rarely an inconvenience.

Building a Tags File

On a piece of source code where all the files are in one directory, a common ctags command might be be:

```
> ctags *.cpp *.h
```

to produce a tags file for vi, or:

```
> ctags –e *.cpp *.h
```

to produce a tags file for Emacs. In many instances, this very simple usage of ctags is all that is required.

On a somewhat larger scale, I was recently looking at the Rekall database application which consists of 350,000 lines of C++ code spread across nearly 900 source files. Starting in the source tree’s root directory, ctags will build a tags file for this block of code with a single line:

```
> ctags –R
```

The –R flag causes ctags to move recursively down the directory structure, finding and tagging all the files which it understands the syntax of. Any which it doesn’t understand it will quietly ignore. Even on my aging PIII-500 Linux box, this process takes less than 10 seconds to produce a tags file with nearly 50,000 entries. This includes the source, the Makefiles, the test scripts and all the other code in the tree.

Once a ctags command has been run, the tags file will be found in the current directory:

```
> ls -l tags
-rw-r--r--    1 derek    users     3423829 2004-03-02 10:45 tags
```

If the ctags utility is run in Emacs mode (with the –e flag) the output file is, by default, called TAGS. This usage of uppercase is expected by the Emacs editor, and is one of the subtle differences between vi mode and Emacs mode.

Sometimes a source tree is a little more complicated. For example, the source tree of the Tcl scripting language is a fairly large chunk of C code, which is complicated by the fact that it contains duplicates of many functions. One single function might be provided for the Unix platform, the Windows platform and again for the Macintosh platform. Depending on what the developer is doing, it might make sense for the tags file to contain just one version (eg. the one the developer is working on).

There are two ways to build a tags file from selected parts of a source tree. The first is to exclude the parts of the tree which are not required:

```
> ctags -R --exclude="win" --exclude="mac"
```

This will exclude from the tags file all the code below the win and mac subdirectories. This simple example works nicely for the Tcl code because it is conveniently laid out. For some other pieces of code it is easier to use a utility like find to locate the required files, then pass them to ctags directly. The ctags –L option allows for a list of files to be passed into the utility in a file. Using the standard input as that file, the Unix-only Tcl example can also be built like this:

```
find . \( -path "./win" -o -path "./mac*" \) -prune -o \
       \( -name "*.[ch]" -a -print \) | ctags -L –
```

That is, find prunes the win and mac parts of the source tree, then prints the names of the C and header files it finds. This list of files is then piped into the ctags utility. For very complicated, multiplatform programs, such as the Linux kernel, it is sometimes necessary to write a script to build the exact tags file required. Such a script would normally use the --append option of ctags to build the tags file piece by piece.

Using Tags in Editors

Any advanced programmer's editor worth its salt can read and utilise tags files. For my examples I'll use the most popular two: vi and Emacs, in their VIM and XEmacs guises.

Both VIM and XEmacs will automatically load a tags file from their current working directory as soon as a tag command is used. If the tags file is somewhere else both editors will need help to find it. In VIM use the "set tags=/path/to/tags" command; in XEmacs use the "visit-tags-table" command (or use the "Set tags table file" option in the Tools->Tags menu).

Once a tags file is loaded, both editors work in much the same way. The underlying concept is a stack of tags which have been followed. When the user asks to jump to a specified symbol, its file and line number are found in the tags file. The current location is noted on the tags stack, then the new location is opened and jumped to. When the user has examined the code at the requested tag, they issue the command which jumps them back to their previous location – i.e. they jump back to the top entry on the stack. This mechanism is intuitive. The user jumps to tag after tag as they work deeper into the code, then jumps back out again.

The VIM command to jump to a named tag is ":ta tag", where "tag" is the symbol required. A common alternative is Ctrl-] which jumps to the symbol currently under the cursor. Ctrl-T is used to pop back out again.

In XEmacs the command Meta-. (that's Meta-<period>) is used to jump to a tag. You are prompted for the name of the tag you want, with the default value being the symbol under the cursor. Meta-* is used to pop back out again.

These commands to jump forwards and backwards between symbols are the only ones needed for 90% of the time. However, both editors have a generous array of commands to manipulate tags, their files and the current state of the stack. These options range from the occasionally useful – such as the ability to see a tag in another window – to the somewhat obscure. See the help pages in your favoured text editor for more information.

Other useful features

Given the range of languages which Exuberant Ctags can understand, it is unlikely that any even vaguely mainstream piece of code can't be tagged using it. Should that event occur, however, Exuberant Ctags has a special regular expression based syntax for finding relevant symbols in program code. With this feature, any code written in any language where the symbols can be extracted using a regular expression can be tagged.

Another feature which will appeal to Emacs users is that if the Exuberant Ctags ctags command is invoked with the name "etags", either because the utility has been renamed, or because a so named soft link has been created to it, ctags automatically runs in Emacs mode.

vi and Emacs are, of course, not the only editors which read and understand tags files. Various other editors support their use, with the best support being offered by the most mature, full featured editors aimed at programmers. The Exuberant Ctags manual page lists the editors which are known to support tags, but you should check the documentation for your favoured editor if it's not on the list. Tag support is neither obscure nor difficult to implement, so many editors include it as a matter of routine these days.

The Exuberant Ctags package comes with a public domain library which can be linked into any existing editor code, and which will enable that editor to use tags. If your favoured editor doesn't include ctags support, email the author. Chances are tag support can be added to most existing editors with very little fuss or effort.

Conclusion

ctags solves the most frequently encountered problem developers experience when working on a large piece of code – finding where everything is. Building tags files is simple, and the files are well supported amongst the foremost programmer's editors. Through its quality and flexibility, the Exuberant Ctags package has taken the prominent position amongst tag generating tools, and is now the standard tags solution.

Anyone working with source trees of any size would be well advised to investigate how ctags and a suitably equipped text editor can make their life easier.

References

Exuberant Ctags project:

http://ctags.sourceforge.net

Exuberant Ctags manual page:

http://ctags.sourceforge.net/ctags.html

VIM tags help page:
http://www.vim.org/htmldoc/usr_29.html

XEmacs tags help page:
http://www.xemacs.org/Documentation/21.5/html/xemacs_24.html#SEC241

The Rekall homepage:
http://www.totalrekall.co.uk

The Tcl scripting language homepage:
http://www.tcl.tk