CSCOPE

NAME

cscope - interactively examine a C program

SYNOPSIS

cscope [**-bCcdehkLlqRTUuV**] [**-F**symfile**]** [**-f**reffile**]** [**-I**incdir**]** [**-i**namefile**]** [**-num**pattern**]** [**-p**n**]** [**-s**dir**]**

DESCRIPTION

cscope is an interactive, screen-oriented tool that allows the user to browse through C source files for specified elements of code.

By default, *cscope* examines the C (.c and .h), lex (.l), and yacc (.y) source files in the current directory. *cscope* may also be invoked for source files named on the command line. In either case, *cscope* searches the standard directories for #include files that it does not find in the current directory. *cscope* uses a symbol cross-reference, cscope.out by default, to locate functions, function calls, macros, variables, and preprocessor symbols in the files.

cscope builds the symbol cross-reference the first time it is used on the source files for the program being browsed. On a subsequent invocation, *cscope* rebuilds the cross-reference only if a source file has changed or the list of source files is different. When the cross-reference is rebuilt, the data for the unchanged files are copied from the old cross-reference, which makes rebuilding faster than the initial build.

OPTIONS

The following options can appear in any combination:

-b
: Build the cross-reference only.

-C
: Ignore letter case when searching.

-c
: Use only ASCII characters in the cross-reference file, that is, do not compress the data.

-d
: Do not update the cross-reference.

-e
: Suppress the <Ctrl>-e command prompt between files.

-F symfile
: Read symbol reference lines from symfile. (A symbol reference file is created by > and >>, and can also be read using the < command, described under ``Issuing Subsequent Requests," below.)

-f reffile
: Use reffile as the cross-reference file name instead of the default cscope.out.

-h
: View the long usage help display.

-I incdir
: Look in incdir (before looking in INCDIR, the standard place for header files, normally /usr/include) for any #include files whose names do not begin with ``/" and that are not specified on the command line or in namefile below. (The #include files may be specified with either double quotes or angle

brackets.) The incdir directory is searched in addition to the current directory (which is searched first) and the standard list (which is searched last). If more than one occurrence of -I appears, the directories are searched in the order they appear on the command line.

-i namefile

Browse through all source files whose names are listed in namefile (file names separated by spaces, tabs, or new-lines) instead of the default (cscope.files). If this option is specified, cscope ignores any files appearing on the command line. The argument namefile can be set to ``-" to accept a list of files from stdio. Filenames in the namefile that contain whitespace have to be enclosed in "double quotes". Inside such quoted filenames, any double-quote and backslash characters have to be escaped by backslashes.

-k

``Kernel Mode", turns off the use of the default include dir (usually /usr/include) when building the database, since kernel source trees generally do not use it.

-L

Do a single search with line-oriented output when used with the -num pattern option.

-l

Line-oriented interface (see ``Line-Oriented Interface" below).

-num pattern

Go to input field num (counting from 0) and find pattern.

-P path

Prepend path to relative file names in a pre-built cross-reference file so you do not have to change to the directory where the cross-reference file was built. This option is only valid with the -d option.

-p n

Display the last n file path components instead of the default (1). Use 0 to not display the file name at all.

-q

Enable fast symbol lookup via an inverted index. This option causes cscope to create 2 more files (default names ``cscope.in.out" and ``cscope.po.out") in addition to the normal database. This allows a faster symbol search algorithm that provides noticeably faster lookup performance for large projects.

-R

Recurse subdirectories for source files.

-s dir

Look in dir for additional source files. This option is ignored if source files are given on the command line.

-T

Use only the first eight characters to match against C symbols. A regular expression containing special characters other than a period (.) will not match any symbol if its minimum length is greater than eight characters.

-U

Check file time stamps. This option will update the time stamp on the database even if no files have changed.

-u

Unconditionally build the cross-reference file (assume that all files have changed).

-V

Print on the first line of screen the version number of cscope.

The -I, -c, -k, -p, -q, and -T options can also be in the cscope.files file.

Requesting the initial search

After the cross-reference is ready, cscope will display this menu:

Find this C symbol:

Find this function definition:
Find functions called by this function:
Find functions calling this function:
Find this text string:

Change this text string:
Find this egrep pattern:
Find this file:
Find files #including this file:

Press the <Up> or <Down> keys repeatedly to move to the desired input field, type the text to search for, and then press the <Return> key.

Issuing subsequent requests

If the search is successful, any of these single-character commands can be used:

0-9a-zA-Z
: Edit the file referenced by the given line number.

<Space>
: Display next set of matching lines.

<Tab>
: Alternate between the menu and the list of matching lines

<Up>
: Move to the previous menu item (if the cursor is in the menu) or move to the previous matching line (if the cursor is in the matching line list.)

<Down>
: Move to the next menu item (if the cursor is in the menu) or move to the next matching line (if the cursor is in the matching line list.)

+
: Display next set of matching lines.

-
: Display previous set of matching lines.

^e
: Edit displayed files in order.

>
: Write the displayed list of lines to a file.

>>
: Append the displayed list of lines to a file.

<
: Read lines from a file that is in symbol reference format (created by > or >>), just like the -F option.

^
: Filter all lines through a shell command and display the resulting lines, replacing the lines that were already there.

|
: Pipe all lines to a shell command and display them without changing them.

At any time these single-character commands can also be used:

<Return>
: Move to next input field.

^n
: Move to next input field.

^p
: Move to previous input field.

^y
: Search with the last text typed.

^b
: Move to previous input field and search pattern.

^f

Move to next input field and search pattern.

^c
Toggle ignore/use letter case when searching. (When ignoring letter case, search for ``FILE'' will match ``File'' and ``file''.)

^r
Rebuild the cross-reference.

!
Start an interactive shell (type ^d to return to cscope).

^l
Redraw the screen.

?
Give help information about cscope commands.

^d
Exit cscope.

NOTE: If the first character of the text to be searched for matches one of the above commands, escape it by typing a (backslash) first.

Substituting new text for old text

After the text to be changed has been typed, cscope will prompt for the new text, and then it will display the lines containing the old text. Select the lines to be changed with these single-character commands:

0-9a-zA-Z
Mark or unmark the line to be changed.

*****
Mark or unmark all displayed lines to be changed.

<Space>
Display next set of lines.

+
Display next set of lines.

-
Display previous set of lines.

a
Mark or unmark all lines to be changed.

^d
Change the marked lines and exit.

<Esc>
Exit without changing the marked lines.

!
Start an interactive shell (type ^d to return to cscope).

^l
Redraw the screen.

?
Give help information about cscope commands.

Special keys

If your terminal has arrow keys that work in vi, you can use them to move around the input fields. The up-arrow key is useful to move to the previous input field instead of using the <Tab> key repeatedly. If you have <CLEAR>, <NEXT>, or <PREV> keys they will act as the ^l, +, and - commands, respectively.

Line-Oriented interface

The -l option lets you use cscope where a screen-oriented interface would not be useful, for example, from another screen-oriented program.

cscope will prompt with >> when it is ready for an input line starting with the field number (counting from 0) immediately followed by the search pattern, for example, ``lmain'' finds the definition of the main function.

If you just want a single search, instead of the -l option use the -L and -num pattern options, and you won't get the >> prompt.

For -l, cscope outputs the number of reference lines cscope: 2 lines

For each reference found, cscope outputs a line consisting of the file name, function name, line number, and line text, separated by spaces, for example, main.c main 161 main(argc, argv)

Note that the editor is not called to display a single reference, unlike the screen-oriented interface.

You can use the c command to toggle ignore/use letter case when searching. (When ignoring letter case, search for ``FILE'' will match ``File'' and ``file''.)

You can use the r command to rebuild the database.

cscope will quit when it detects end-of-file, or when the first character of an input line is ``^d'' or ``q''.

ENVIRONMENT VARIABLES

CSCOPE_EDITOR
: Overrides the EDITOR and VIEWER variables. Use this if you wish to use a different editor with cscope than that specified by your EDITOR/VIEWER variables.

CSCOPE_LINEFLAG
: Format of the line number flag for your editor. By default, cscope invokes your editor via the equivalent of ``editor +N file'', where ``N'' is the line number that the editor should jump to. This format is used by both emacs and vi. If your editor needs something different, specify it in this variable, with ``%s'' as a placeholder for the line number. Ex: if your editor needs to be invoked as ``editor -#103 file'' to go to line 103, set this variable to ``-#%s''.

CSCOPE_LINEFLAG_AFTER_FILE
: Set this variable to ``yes'' if your editor needs to be invoked with the line number option after the filename to be edited. To continue the example from CSCOPE_LINEFLAG, above: if your editor needs to see ``editor file -#number'', set this environment variable. Users of most standard editors (vi, emacs) do not need to set this variable.

EDITOR
: Preferred editor, which defaults to vi.

HOME
: Home directory, which is automatically set at login.

INCLUDEDIRS
: Colon-separated list of directories to search for #include files.

SHELL
: Preferred shell, which defaults to sh.

SOURCEDIRS
: Colon-separated list of directories to search for additional source files.

TERM
: Terminal type, which must be a screen terminal.

TERMINFO
: Terminal information directory full path name. If your terminal is not in the standard terminfo directory, see curses and terminfo for how to make your own terminal description.

TMPDIR
: Temporary file directory, which defaults to /var/tmp.

VIEWER
: Preferred file display program (such as less), which overrides EDITOR (see above).

VPATH
: A colon-separated list of directories, each of which has the same directory structure below it. If VPATH is set, cscope searches for source files in the directories specified; if it is not set, cscope searches only in the current directory.

FILES

cscope.files
Default files containing -I, -p, -q, and -T options and the list of source files (overridden by the -i option).
cscope.out
Symbol cross-reference file (overridden by the -f option), which is put in the home directory if it cannot be created in the current directory.
cscope.in.out
cscope.po.out
Default files containing the inverted index used for quick symbol searching (-q option). If you use the -f option to rename the cross-reference file (so it's not cscope.out), the names for these inverted index files will be created by adding
.in and .po to the name you supply with -f. For example, if you indicated -f xyz, then these files would be named xyz.in and xyz.po.
INCDIR
Standard directory for #include files (usually /usr/include).

Notices

cscope recognizes function definitions of the form:

fname blank (args) white arg_decs white {
where:
fname is the function name
blank
is zero or more spaces or tabs, not including newlines
args
is any string that does not contain a ``'' or a newline
white
is zero or more spaces, tabs, or newlines
arg_decs
are zero or more argument declarations (arg_decs may include comments and white space)

It is not necessary for a function declaration to start at the beginning of a line. The return type may precede the function name; cscope will still recognize the declaration. Function definitions that deviate from this form will not be recognized by cscope.

The ``Function'' column of the search output for the menu option Find functions called by this function: input field will only display the first function called in the line, that is, for this function

```
e()
{
        return (f() + g());
}
```

the display would be

```
Functions called by this function: e
File Function Line
a.c f 3 return(f() + g());
```

Occasionally, a function definition or call may not be recognized because of braces inside #if statements. Similarly, the use of a variable may be incorrectly recognized as a definition.

A **typedef** name preceding a preprocessor statement will be incorrectly recognized as a global definition, for example,

```
LDFILE  *
#if AR16WR
```

Preprocessor statements can also prevent the recognition of a global definition, for example,

```
char flag
#ifdef ALLOCATE_STORAGE
    = -1
#endif
;
```

A function declaration inside a function is incorrectly recognized as a function call, for example,

```
f()
{
      void g();
}
```

is incorrectly recognized as a call to g.

cscope recognizes C++ classes by looking for the class keyword, but doesn't recognize that a struct is also a class, so it doesn't recognize inline member function definitions in a structure. It also doesn't expect the class keyword in a *typedef* , so it incorrectly recognizes X as a definition in

```
typedef class X  *  Y;
```

It also doesn't recognize operator function definitions

```
Bool Feature::operator==(const Feature & other)
{
  ...
}
```

Nor does it recognize function definitions with a function pointer argument

```
ParseTable::Recognize(int startState, char *pattern,
  int finishState, void (*FinalAction)(char *))
{
  ...
}
```

Index
