ctags - Generates "tags" and (optionally) "refs" files

SYNOPSIS

ctags [**-D** *word*] [**-FBNitvshlpxra**] *files...*

DESCRIPTION

This page describes the *elvis* version of *ctags*.

ctags generates the "tags" and "refs" files from a group of C source files. The "tags" file is used by Elvis' ":tag" command, **^]** command, and **-t** option. The "refs" file is sometimes used by the **ref(1)** program.

Each C source file is scanned for #define statements and global function definitions. The name of the macro or function becomes the name of a tag. For each tag, a line is added to the "tags" file.

The filenames list will typically be the names of all C source files in the current directory, like this:

```
$ ctags *.c *.h
```

OPTIONS

If no options are given, then *ctags* acts as though the **-l** **-i** **-t** **-v** and **-s** option flags were given. If you want to omit those options, you can do so by explicitly giving a harmless option such as **-F**.

-D*word* This causes elvis to ignore any instance of *word* in your source code. This is handy if you're using a macro for conditionally declaring the arguments to functions, in order to make your code be backward-compatible with older K&R C compilers. *ctags* always ignores "P_" and "__P"; the **-D***word* flag allows you to make it ignore a third word.

-F Enclose regular expressions in slashes (/regexp/) which will cause **elvis(1)** to search form the top of the file. This is the default.

-B Enclose the regular expressions in question marks (?regexp?) so **elvis(1)** will search backward from the bottom of the file. The search direction rarely matters; this option exists mostly for compatibility with earlier versions of ctags.

-N This causes *ctags* to use line numbers for all tags. Without this flag, it would use numbers for #define'ed macros, and regular expressions for anything else.

being inline, __inline, or __inline__.

-t Include typedefs. A tag will be generated for each user-defined type. Also tags will be generated for struct and enum names. Types are considered to be global if they are defined in a header file, and static if they are defined in a C source file.

-v Include variable declarations. A tag will be generated for each variable, except for those that are declared inside the body of a function.

-s Include static tags. *ctags* will normally put global tags in the "tags" file, and silently ignore the static tags. This flag causes both global and static tags to be added.

-e Include extern tags. *ctags* will normally ignore extern declarations of functions or variables; that's handy when generating tags for your own programs. A tags file for the extern declarations in the system's standard header files can be a very handy resource, so this **-e** flag was created.

-h Add hints that may help elvis handle overloaded tags better. The resulting tags file may be unreadable by programs other than elvis, though.

-l Add "ln" line number hints. This implies **-h**, since it would be pointless if hints weren't allowed. The "ln" hints are used by **elvis(1)** to make its "showtag" option work much faster.

-p Write parsing information to stdout. This is intended mainly as an aid to debugging the *ctags* command itself. If *ctags* doesn't generate all of the tags that you expect it to, then try studying the **-p** output to determine what syntax feature is tripping it up.

-x Generate a human-readable tag list instead of a "tags" file. The list is written to stdout. Each line contains a tag name, the line number and file name where the tag is defined, and the text of that line.

-r This causes *ctags* to generate both "tags" and "refs". Without **-r**, it would only generate "tags".

-a Append to "tags", and maybe "refs". Normally, *ctags* overwrites these files each time it is invoked. This flag is useful when you have too split the arguments among several invocations. **This may result in an unsorted tags file.**

FORMAT OF THE TAGS FILE

The "tags" file is a text file. Each line stores the attributes of a single tag. The basic format of a line is:

- the name of the tag
- a tab character
- the name of the file containing the tag
- a tab character
- the tag's address within that file

The tag address may be given as either line number (a string of digits), or a regular expression using ex/vi's "nomagic" syntax, delimited by either slashes or question marks. Regular expressions are allowed to contain tab characters.

The authors of elvis, vim, and "exuberant" ctags have agreed on a standard format for adding additional attributes to tags. In this format, the first three fields of all tags are identical to the traditional format, except that a semicolon-doublequote character pair is appended to the tag address field, with the extra attributes appearing after that.

The semicolon-doublequote character pair is present because it has the surprising side-effect of making the original ex/vi ignore the remainder of the line, thus allowing the original ex/vi to read new-format tags files. The original ex/vi will simply ignore the extra attributes.

Any additional attributes are appended to the tag's line. They may be appended in any order. Each attribute will use the following format:

- a tab character
- the name of the attribute
- a colon character, ':'
- the value of the attribute.

Note that each additional attribute has an explicit name. Different tags files may use totally different names for additional attributes, and even within a single file, most tags will use only a subset of the possible attributes. This version of ctags uses the following names:

file   This attribute is used to mark static tags -- i.e., tags for C/C++ functions or variables whose scope is limited to the function in which they are defined. The value is the name of the file where it is defined, except that if the file is the same

class This is used to mark member functions of C++ classes. The value is the class name. However, currently ctags doesn't do a very good job of detecting whether a function is a member function or not.

kind   This attribute's value is a single letter, indicating the lexical type of the tagged identifier: **f** for a function, **t** for a typedef, **s** for a struct tag, **u** for a union tag, **v** for a variable, **d** for a macro definition, or **x** for an extern declaration.

Note that in the tags file, the "kind:" label is omitted, for the sake of compactness.

ln   This gives the line number where the tag was defined. It is redundant, but it is still somewhat useful because it allows **elvis(1)**'s "showtag" option to work faster.

The values can only contain tabs if those tabs are converted to the '\t' (backslash-t) notation. Similarly, a newline, carriage return, or literal backslash can be given as '\n', '\r', or '\\' respectively. For MS-DOS file names, this means the names must use double backslashes. Space characters don't require any special encoding. (This doesn't apply to file names in the *tagfile* field, where names can be given without any special encoding. It only applies to file names in extra fields.)

As a special case, if an extra attribute contains no ':' to delimit the name from the value, then the attribute string is assumed to be the value of an attribute named "kind". Usually this will be a single letter indicating what type of token the tag represents -- 'f' for function, 'v' for variable, and so on.

Here's an example of a new-format tag:

```
bar  foo.c    /^void Foo::bar(int zot)$/;"  class:Foo
```

The tagname is "bar", to match its function's name. The tagfile is "foo.c". The tagaddress is a regular expression containing the whole definition line. Note that a semicolon-doublequote character pair has been appended to the tagaddress. There is only one additional attribute, with the name "class" and the value "Foo".

FILES

tags   A cross-reference that lists each tag name, the name of the source file that contains it, and a way to locate a particular line in the source file.

file can be useful, for example, when licensing restrictions prevent you from making the source code to the standard C library readable by everybody, but you still want everybody to know what arguments the library functions need.

BUGS

ctags is sensitive to indenting and line breaks. Consequently, it might not discover all of the tags in a file that is formatted in an unusual way.

The **-a** flag causes tag files to be appended, but not necessarily sorted. Some programs expect tags files to be sorted, and will misbehave if they aren't. Also, the new format allows a "!_TAG_FILE_SORTED" marker near the top of the file to indicate whether the file is sorted, but that might not be accurate after new tags are appended to the file. Consequently, you should avoid the use of **-a**.

The new standard doesn't specify how overloaded operators are to be labelled. If your C++ source contains a definition of operator+=(), then this version of *ctags* will store a tag named "operator+=". Other versions of ctags could simply use the name "+=".

SEE ALSO

elvis(1), ref(1)

AUTHOR

Steve Kirkendall
kirkenda@cs.pdx.edu