Donate

easytags.vim Automated tag generation and syntax highlighting in Vim

Ohloh: Mostly written in Vim Script, 1368 lines of code, 4 contributors in last 12 months
GitHub: Watched by 488 people, most recent update was 124 months ago
Vim Online: Rated 549, downloaded 10728 times

Vim has long been my favorite text editor and combined with Exuberant Ctags it has the potential to provide most of what I expect from an integrated development environment. Exuberant Ctags is the latest incarnation of a family of computer programs that scan source code files to create an index of identifiers (tags) and where they are defined. Vim uses this index (a so-called tags file) to enable you to jump to the definition of any identifier using the Control-] mapping.

When you're familiar with integrated development environments you may recognize this feature as "Go-to definition". One advantage of the combination of Vim and Exuberant Ctags over integrated development environments is that Vim supports syntax highlighting for over 500 file types (!) and Exuberant Ctags can generate tags for over 40 file types as well…

There's just one problem: You have to manually keep your tags files up-to-date and this turns out to be a royal pain in the ass! So I set out to write a Vim plug-in that would do this boring work for me. When I finished the plug-in's basic functionality (one automatic command and a call to system() later) I became interested in dynamic syntax highlighting, so I added that as well to see if it would work — surprisingly well I'm happy to report!

Installation

Please refer to the installation instructions available on GitHub.

Now try it out: Edit any file type supported by Exuberant Ctags and within ten seconds the plug-in should create/update your tags file (`~/.vimtags` on UNIX, `~/_vimtags` on Windows) with the tags defined in the file you just edited! This means that whatever file you're editing in

Vim (as long as it's on the local file system), tags will always be available by the time you need them!

Additionally if the file you just opened is an AWK, C#, C, C++, Objective-C, Java, Lua, Perl, PHP, Python, Ruby, Shell, Tcl or Vim source file you should also notice that the function and/or type names defined in the file have been syntax highlighted.

The `easytags.vim` plug-in is intended to work automatically once it's installed, but if you want to change how it works there are several options you can change and commands you can execute from your own mappings and/or automatic commands. These are all documented below.

Note that if the plug-in warns you `ctags` isn't installed you'll have to download it from its homepage, or if you're running Debian/Ubuntu you can install it by executing the following shell command:

```
$ sudo apt-get install exuberant-ctags
```

A note about Windows

On Windows the system() function used by `easytags.vim` causes a command prompt window to pop up while Exuberant Ctags is executing. If this bothers you then you can install my shell.vim plug-in which includes a DLL that works around this issue. Once you've installed both plug-ins it should work out of the box! Please let me know if this doesn't work for you.

Commands

The `:UpdateTags` command

This command executes Exuberant Ctags from inside Vim to update the global tags file defined by `g:easytags_file`. When no arguments are given the tags for the current file are updated, otherwise the arguments are passed on to `ctags`. For example when you execute the Vim command `:UpdateTags -R ~/.vim` (or `:UpdateTags -R ~\vimfiles` on Windows) the plug-in will execute `ctags -R ~/.vim` for you (with some additional arguments, see the troubleshooting section "`:HighlightTags` only works for the tags file created by `:UpdateTags`" for more information).

When you execute this command like `:UpdateTags!` (including the bang!) then all tags whose files are missing will be filtered from the global tags file.

Note that this command will be executed automatically every once in a while, assuming you haven't changed `g:easytags_on_cursorhold`.

The `:HighlightTags` command

When you execute this command while editing one of the supported file types (see above) the relevant tags in the current file are highlighted. The tags to highlight are gathered from all tags files known to Vim (through the 'tags' option).

Note that this command will be executed automatically every once in a while, assuming you haven't changed `g:easytags_on_cursorhold`.

Options

The easytags plug-in should work out of the box but if you don't like the default configuration you can change how it works by setting the variables documented below. Most of these variables can also be changed for specific files by setting a buffer local variable instead of the global variable. For example to disable automatic highlighting (enabled by default) only in Python files you can add the following line to your vimrc script:

```
:autocmd FileType python let b:easytags_auto_highlight = 0
```

Note that buffer local variables always override global variables, so if you want to undo this for a specific file you have to use :unlet:

```
:unlet b:easytags_auto_highlight
```

The `g:easytags_cmd` option

The plug-in will try to determine the location where Exuberant Ctags is installed on its own but this might not always work because any given executable named `ctags` in your `$PATH` might not in fact be Exuberant Ctags but some older, more primitive `ctags` implementation which doesn't support the same command line options and thus breaks the easytags plug-in. If this is the case you can set the global variable `g:easytags_cmd` to the location where you've installed Exuberant Ctags, e.g.:

```
:let g:easytags_cmd = '/usr/local/bin/ctags'
```

If you rely entirely on language-specific configuration and don't have a general ctags program, set this to the empty string.

The `g:easytags_opts` option

If you need to pass custom command line option(s) to the program specified by `g:easytags_cmd` you can set this option to a list of strings to be passed to Exuberant Ctags. Make sure to only use options that are valid in any context, for example the concatenation of `g:easytags_cmd`, `g:easytags_opts` and `--list-languages` should work as expected. Here's an example:

```
:let g:easytags_opts = ['--options=$VIM\ctags\ctags.cnf']
```

The example above (based on issue 98) overrides the location of Exuberant Ctags' configuration file. As you can see the command line option(s) may contain environment variables, these will be expanded before passing the options to Exuberant Ctags (to make sure it works in all environments).

The `g:easytags_async` option

By default vim-easytags runs Exuberant Ctags and updates your tags file in the foreground, blocking Vim in the process. As your tags files get larger this becomes more annoying. It has been the number one complaint about vim-easytags since I published the first release online.

In version 3.5 of the vim-easytags plug-in support for asynchronous tags file updates was added. It's not enabled by default yet because I want to make sure I'm not breaking the plug-in for the majority of users. However after I've gathered some feedback I definitely want to make this the default mode.

By setting this option to true (1) you enable asynchronous tags file updates. Good luck! ;-)

Note that asynchronous updates on Windows currently require the installation of my vim-shell plug-in (for obscure technical reasons that I want to fix but don't know how yet).

The `g:easytags_syntax_keyword` option

When you look into how the dynamic syntax highlighting in the vim-easytags plug-in works you'll realize that vim-easytags is really abusing Vim's syntax highlighting engine. This can cause Vim to slow down to a crawl, depending on how big your tags files are. To make things worse in Vim 7.4 a new regex engine was introduced which exacerbates the problem (the patterns generated by vim-easytags bring out the worst of the new regex engine).

Since version 3.6 the vim-easytags plug-in tries to squeeze as much performance as possible out of Vim by using keyword highlighting where this is possible without sacrificing accuracy. If your Vim's syntax highlighting is still too slow you can add the following to your vimrc script:

```
let g:easytags_syntax_keyword = 'always'
```

The default value of this option is 'auto' which means to use keyword highlighting where this is possible without sacrificing accuracy. By changing it to 'always' you're telling vim-easytags to sacrifice accuracy in order to gain performance. Try it out and see what works best for you.

Please note that right now this 'feature' is not integrated with the "accelerated Python syntax highlighting" feature, because I'm considering ripping that out and replacing it with a *fast* Vim script implementation.

The `g:easytags_languages` option

Exuberant Ctags supports many languages and can be extended via regular expression patterns, but for some languages separate tools with ctags-compatible output exist (e.g. jsctags for Javascript). To use these, the executable and its arguments must be configured:

```
let g:easytags languages = {
\   'language': {
\     'cmd': g:easytags_cmd,
\       'args': [],
\       'fileoutput opt': '-f',
\       'stdout opt': '-f-',
\       'recurse_flag': '-R'
\   }
\}
```

Each key is a special language definition. The key is a Vim file type in lowercase. The above snippet shows the defaults; you only need to specify options that differ.

The `g:easytags_file` option

As mentioned above the plug-in will store your tags in `~/.vimtags` on UNIX and `~/_vimtags` on Windows. To change the location of this file, set the global variable `g:easytags_file`, e.g.:

```
:let g:easytags_file = '~/.vim/tags'
```

A leading `~` in the `g:easytags_file` variable is expanded to your current home directory (`$HOME` on UNIX, `%USERPROFILE%` on Windows).

The `g:easytags_dynamic_files` option

By default `:UpdateTags` only writes to the global tags file, but it can be configured to look for project specific tags files by adding the following lines to your vimrc script:

```
:set tags=./tags;
:let g:easytags_dynamic_files = 1
```

You can change the name of the tags file, the important thing is that it's relative to your working directory or the buffer (using a leading `./`). When `g:easytags_dynamic_files` is set to 1 the easytags plug-in will write to the first existing tags file seen by Vim (based on the 'tags' option). In other words: If a project specific tags file is found it will be used, otherwise the plug-in falls back to the global tags file (or a file type specific tags file).

If you set `g:easytags_dynamic_files` to 2 the easytags plug-in will automatically create a project specific tags file based on the first name in the 'tags' option. In this mode the global tags file and/or file type specific tags files are only used for directories where you don't have write permissions.

When you set `g:easytags_dynamic_files` to 2 new tags files are created in the same directory as the file you're editing. If you want the tags files to be created in your working directory instead then change Vim's 'cpoptions' option to include the lowercase letter 'd'.

The 'tags' option is reevaluated each time the plug-in runs, so which tags file is selected can differ depending on the buffer and working directory.

The `g:easytags_by_filetype` option

By default all tags are stored in a global tags file. When the tags file grows beyond a certain size Vim will be slowed down by the easytags plug-in because it has to read and process a large number of tags very frequently.

To avoid this problem you can set `g:easytags_by_filetype` to the path of an existing directory. The easytags plug-in will create separate tags files for each file type in the configured directory. These tags files are automatically registered by the easytags plug-in when the file type of a buffer is set.

Note that the `g:easytags_dynamic_files` option takes precedence over this option.

If you already have a global tags file you can create file type specific tags files from the global tags file using the command `:TagsByFileType`.

The `g:easytags_events` option

This option can be used to customize the events that trigger the automatic updating and highlighting performed by the easytags plug-in. The `g:easytags_always_enabled` and `g:easytags_on_cursorhold` options are more user friendly but limited ways to accomplish the same thing.

Here's an example: Say you want the easytags plug-in to automatically update & highlight tags for the current file right after you save the file. You can accomplish this by adding the following line to your [vimrc script](http://vimdoc.sourceforge.net/htmldoc/starting.html#vimrc):

```
:let g:easytags_events = ['BufWritePost']
```

Note that if you set `g:easytags_events` in your vimrc script, the values of the options `g:easytags_always_enabled` and `g:easytags_on_cursorhold` will be ignored completely.

The `g:easytags_always_enabled` option

By default the plug-in automatically generates and highlights tags when you stop typing for a few seconds (this works using the CursorHold automatic command). This means that when you edit a file, the dynamic highlighting won't appear until you pause for a moment. If you don't like this you can configure the plug-in to always enable dynamic highlighting:

```
:let g:easytags_always_enabled = 1
```

Be warned that after setting this option you'll probably notice why it's disabled by default: Every time you edit a file in Vim, the plug-in will first run Exuberant Ctags and then highlight the tags, and this slows Vim down quite a lot. To make this less of a problem you can use the `g:easytags_async` option.

Note: If you change this option it won't apply until you restart Vim, so you'll have to set this option in your vimrc script.

The `g:easytags_on_cursorhold` option

As I explained above the plug-in by default doesn't update or highlight your tags until you stop typing for a moment. The plug-in tries hard to do the least amount of work possible in this break but it might still interrupt your workflow. If it does you can disable the periodic update:

```
:let g:easytags_on_cursorhold = 0
```

Note: Like the `g:easytags_always_enabled` option, if you change this option it won't apply until you restart Vim, so you'll have to set this option in your vimrc script.

The `g:easytags_updatetime_min` option

Vim's 'updatetime' option controls how often the easytags plug-in is automatically executed. A lot of popular Vim plug-ins manipulate this option to control how often they are called. Unfortunately some of those plug-ins set 'updatetime' to a very low value (less than a second) and this can break the easytags plug-in. Because of this the easytags plug-in compensates by keeping track of when it was last executed.

The default value of Vim's 'updatetime option *and* the `g:easytags_updatetime_min` option is 4000 milliseconds (4 seconds).

If you know what you're doing and you really want the easytags plug-in to be executed more than once every 4 seconds you can lower the minimum acceptable updatetime by setting `g:easytags_updatetime_min` to the number of milliseconds (an integer).

Note that although `g:easytags_updatetime_min` counts in milliseconds, the easytags plug-in does not support subsecond granularity because it is limited by Vim's localtime()

function which has one-second resolution.

The `g:easytags_auto_update` option

By default the plug-in automatically updates and highlights your tags when you stop typing for a moment. If you want to disable automatic updating while keeping automatic highlighting enabled you can set this option to false:

```
:let g:easytags_auto_update = 0
```

This disables all *automatic* tags file updates (regardless of how they were enabled) where automatic means *initiated by a Vim automatic command*.

The `g:easytags_auto_highlight` option

By default the plug-in automatically updates and highlights your tags when you stop typing for a moment. If you want to disable automatic highlighting while keeping automatic updating enabled you can set this option to false:

```
:let g:easytags_auto_highlight = 0
```

This disables all *automatic* tags highlighting (regardless of how it was enabled) where automatic means *initiated by a Vim automatic command*.

The `g:easytags_autorecurse` option

When the `:UpdateTags` command is executed automatically or without arguments, it defaults to updating just the tags for the current file. If you'd rather have it recursively scan everything below the directory of the current file then set this option to true (1):

```
:let g:easytags_autorecurse = 1
```

You have to explicitly enable this option because it should only be used while navigating around small directory trees. Imagine always having this option enabled and then having to edit a file in e.g. the root of your home directory: The `easytags.vim` plug-in would freeze Vim for a long time while you'd have to wait for Exuberant Ctags to scan thousands of files…

Note that when you enable this option the `easytags.vim` plug-in might ignore other options like `g:easytags_resolve_links`. This is an implementation detail which I intend to fix.

The `g:easytags_include_members` option

Exuberant Ctags knows how to generate tags for struct/class members in C++ and Java source code but doesn't do so by default because it can more than double the size of your tags files, thus taking much longer to read/write the tags file. When you enable the `g:easytags_include_members` option from your vimrc script (before the `easytags.vim` plug-in is loaded):

```
:let g:easytags_include_members = 1
```

Exuberant Ctags will be instructed to include struct/class members using the `--extra=+q` command line argument and the `easytags.vim` plug-in will highlight them using the `cMember` highlighting group. Because most color schemes don't distinguish the Identifier and Type highlighting groups all members will now probably look like type definitions. You can change that by executing either of the following Vim commands (from your vimrc script, a file type plug-in, etc.):

```
" If you like one of the existing styles you can link them:
highlight link cMember Special

" You can also define your own style if you want:
highlight cMember gui=italic
```

The `g:easytags_resolve_links` option

UNIX has symbolic links and hard links, both of which conflict with the concept of having one unique location for every identifier. With regards to hard links there's not much anyone can do, but because I use symbolic links quite a lot I've added this option. It's disabled by default since it has a small performance impact and might not do what unknowing users expect it to: When you enable this option the plug-in will resolve symbolic links in pathnames, which means your tags file will only contain entries with canonical pathnames. To enable this option (which I strongly suggest doing when you run UNIX and use symbolic links) execute the following Vim command:

```
:let g:easytags_resolve_links = 1
```

The `g:easytags_suppress_ctags_warning` option

If this is set and not false, it will suppress the warning on startup if ctags is not found or not recent enough.

```
:let g:easytags_suppress_ctags_warning = 1
```

The `g:easytags_suppress_report` option

If this is set and not false, it will suppress the report displayed on tag updates.

```
:let g:easytags_suppress_report = 1
```

Customizing the easytags plug-in

Advanced users may wish to customize how the easytags plug-in works beyond the point of changing configuration defaults. This section contains some hints about this. If you have suggestions, please feel free to submit them.

Passing custom command line arguments to Exuberant Ctags

You may want to run Exuberant Ctags with specific command line options, for example the code_complete plug-in requires the signature field to be present. To do this you can create a configuration file for Exuberant Ctags, e.g. `~/.ctags` on UNIX or `%USERPROFILE%\ctags.cnf` on Windows. The file should contain one command line option per line. See the Exuberant Ctags manual for details.

Update & highlight tags immediately after save

By default the easytags plug-in automatically updates & highlights tags for the current file after several seconds of inactivity. This is done to prevent the easytags plug-in from interrupting your workflow.

If you want the easytags plug-in to automatically update & highlight tags for the current file right after you save the file, you can add the following line to your vimrc script:

```
:let g:easytags_events = ['BufWritePost']
```

How to customize the highlighting colors?

The easytags plug-in defines new highlighting groups for dynamically highlighted tags. These groups are linked to Vim's default groups so that they're colored out of the box, but if you want you can change the styles. To do so use a `highlight` command such as the ones given a few paragraphs back. Of course you'll need to change the group name. Here are the group names used by the easytags plug-in:

- **AWK**: `awkFunctionTag`
- **C#:** `csClassOrStructTag`, `csMethodTag`
- **C, C++, Objective C:** `cTypeTag`, `cEnumTag`, `cPreProcTag`, `cFunctionTag`, `cMemberTag`
- **Java:** `javaClassTag`, `javaInterfaceTag`, `javaMethodTag`
- **Lua:** `luaFuncTag`
- **Perl:** `perlFunctionTag`
- **PHP:** `phpFunctionsTag`, `phpClassesTag`
- **Python:** `pythonFunctionTag`, `pythonMethodTag`, `pythonClassTag`
- **Ruby:** `rubyModuleNameTag`, `rubyClassNameTag`, `rubyMethodNameTag`
- **Shell**: `shFunctionTag`
- **Tcl**: `tclCommandTag`
- **Vim:** `vimAutoGroupTag`, `vimCommandTag`, `vimFuncNameTag`, `vimScriptFuncNameTag`

As you can see each of these names ends in `Tag` to avoid conflicts with the syntax modes shipped with Vim. And about the singular/plural confusion: I've tried to match the existing highlighting groups defined by popular syntax modes (except of course for the `Tag` suffix).

Faster syntax highlighting using Python

The Vim script implementation of dynamic syntax highlighting is quite slow on large tags files. When the Python Interface to Vim is enabled the easytags plug-in will therefor automatically use a Python script that performs dynamic syntax highlighting about twice as fast as the Vim script implementation. The following options are available to change the default configuration.

The `g:easytags_python_enabled` option

To disable the Python implementation of dynamic syntax highlighting you can set this option to false (0).

The `g:easytags_python_script` option

This option defines the pathname of the script that contains the Python implementation of dynamic syntax highlighting.

Troubleshooting

vim-easytags is slow!

Yes, I know. I'm trying to make it faster but that's far from trivial. In the process of trying to speed up vim-easytags I've added reporting of elapsed time in several ways. If Vim seems very slow and you suspect this plug-in might be the one to blame, increase Vim's verbosity level:

```
:set vbs=1
```

Every time the plug-in executes it will time how long the execution takes and add the results to Vim's message history, which you can view by executing the :messages command. If you want a more fine grained impression of the time spent by vim-easytags on various operations you can call the `xolox#easytags#why_so_slow()` function:

```
:call xolox#easytags#why_so_slow()
easytags.vim 3.6.4: Timings since you started Vim:
 - 0.094937 seconds updating tags
 - 1.850201 seconds highlighting tags
 - 0.035167 seconds highlighting tags using ':syntax match')
 - 0.493910 seconds highlighting tags using ':syntax keyword')
 - 0.413160 seconds filtering tags for highlighting (stage 1)
 - 0.141747 seconds filtering tags for highlighting (stage 2)
```

`:HighlightTags` only works for the tags file created by `:UpdateTags`

If you want to create tags files and have their tags highlighted by the `easytags.vim` plug-in then you'll have to create the tags file with certain arguments to Exuberant Ctags:

```
$ ctags --fields=+l --c-kinds=+p --c++-kinds=+p ...
```

The `--fields=+l` argument makes sure that Exuberant Ctags includes a `language:...` property with each entry in the tags file. This is required by the `:HighlightTags` command so it can filter tags by their file type. The other two arguments make sure Exuberant Ctags generates tags for function prototypes in C/C++ source code.

If you have the `g:easytags_include_members` option enabled (its off by default) then you'll also need to add the `--extra=+q` argument so that Exuberant Ctags generates tags for structure/class members.

The plug-in complains that Exuberant Ctags isn't installed

After a Mac OS X user found out the hard way that the `ctags` executable isn't always Exuberant Ctags and we spend a few hours debugging the problem I added proper version detection: The plug-in executes `ctags --version` when Vim is started to verify that Exuberant Ctags 5.5 or newer is installed. If it isn't Vim will show the following message on startup:

```
 easytags.vim: Plug-in not loaded because Exuberant Ctags isn
Please download & install Exuberant Ctags from http://ctags.
```

If the installed Exuberant Ctags version is too old the plug-in will complain:

```
 easytags.vim: Plug-in not loaded because Exuberant Ctags 5.5
or newer is required while you have version %s installed!
```

If you have the right version of Exuberant Ctags installed but the plug-in still complains, try executing the following command from inside Vim:

```
:!which ctags
```

If this doesn't print the location where you installed Exuberant Ctags it means your system already had a `ctags` executable but it isn't compatible with Exuberant Ctags 5.5 and you'll need to set the `g:easytags_cmd` option (see above) so the plug-in knows which `ctags` to run.

Vim locks up while the plug-in is running

Once or twice now in several years I've experienced Exuberant Ctags getting into an infinite loop when given garbage input. In my case this happened by accident a few days ago :-|. Because my plug-in executes `ctags` in the foreground this will block Vim indefinitely! If this happens you might be able to kill `ctags` by pressing Control-C but if that doesn't work you can also kill it without stopping Vim using a task manager or the `pkill` command (available on most UNIX systems):

```
$ pkill -KILL ctags
```

Failed to highlight tags because pattern is too big!

If the `easytags.vim` plug-in fails to highlight your tags and the error message mentions that the pattern is too big, your tags file has grown too large for Vim to be able to highlight all tagged identifiers! I've had this happen to me with 50 KB patterns because I added most of the headers in `/usr/include/` to my tags file. Internally Vim raises the error E339: Pattern too long and unfortunately the only way to avoid this problem once it occurs is to reduce the number of tagged identifiers…

In my case the solution was to move most of the tags from `/usr/include/` over to project specific tags files which are automatically loaded by Vim when I edit files in different projects because I've set the 'tags' option as follows:

```
:set tags=./.tags;,~/.vimtags
```

Once you've executed the above command, Vim will automatically look for a file named `.tags` in the directory of the current file. Because of the `;` Vim also recurses upwards so that you can nest files arbitrarily deep under your project directories.

The plug-in doesn't seem to work in Cygwin

If you want to use the plug-in with Vim under Cygwin, you need to have the Cygwin version of Ctags installed instead of the Windows version (thanks to Alex Zuroff for reporting this!).

Contact

If you have questions, bug reports, suggestions, etc. the author can be contacted at peter@peterodding.com. The latest version is available at http://peterodding.com/code/vim/easytags/ and http://github.com/xolox/vim-easytags. If you like this plug-in please vote for it on Vim Online.

License

Thanks go out to everyone who has helped to improve the vim-easytags plug-in (whether through pull requests, bug reports or personal e-mails).

Last updated 124 months ago.