# Ctags vim cheat sheet

CD to the root directory of your Linux kernel code: CD /cse451/user/project1/linux-2.6.13.2/ Run Etags (Ctags for Emacs) on the kernel to generate the TAGS file. etags will overflow its stack if it is called recursively over the entire core, so we use the command to find to find all the .c, .h, and . S (assembly) files in the kernel, then say etags to add the tags in these files to the TAGS file. For Linux 2.6.13, it should only take about a minute: find . -type f-iname O. [chS] » | xargs etags -a Note You can see messages like Warning: can't open the source file '...': Permission denied while etags is building the tag file. These warnings can be ignored. Open any Linux source file in Emacs and use the following basic commands: Action M-keyboard command. Jump to the label under the M-cursor. Look for a particular C-u M-beacon. Find the following definition of the latest M-Pop tag at the place where you previously invoked M-. Important The first time you run an Etags command within Emacs, you may need to specify the location of your TAGS file (i.e. /cse451/.../linux-2.6.13.2/TAGS). Say yes when asked to load the really large tags file. The first command is probably the one you'll use most often: it switches to the definition of the label (function name, structure name, variable name, or just about anything). The second command can be used to search for any tag in the TAGS file, regardless of which file you are currently viewing. Sometimes Etags will find several definitions for a given tag; When this is the case, use the third command to jump through possible definitions until you find the one you want. Finally, use the fourth command to jump back into the stack label. You'll probably find that for some tags (common structures, for example), Etags finds hundreds or thousands of uses in the code, and jumping through them (with the third command above) to try to find the original definition is useless. In this case, you can execute the following two commands to list all the uses of a given: M-x tags-apropos 'tags'apropos (regexp): This will display a list of tag definitions in another buffer. Switch to the new buffer (C-x o), scroll through the list of definitions to the one you want, and then tap Enter to open the file. When you're done, instead of jumping back into the tag stack, close the new buffer (C-x k). To go back to your original tampon and extend it, use C-x o to to it, then C-x 1 to extend. Note Even the list of all definitions given by tags-apropos may be too large to find the definition you are looking for. This is mainly a problem for structs (inode struct, for example) that are used frequently in the nucleus. You should always find Etags useful for jumping to function definitions and less commonly used structs. Ctags for Vim seems to be doing a better job of separating definitions of uses in its tags file, so it's less of a problem for Vim; for Emacs, it's a RET/RET other ways to mitigate this problem (see this page, for example). Alternatively, you can use cscope to find function and structure definitions, or simply use the third step of the Vim instructions below. Instantly share code, notes and excerpts. You can't do this right now. You logged in with another tab or window. Recharge to refresh your session. You logged into another tab or window. Recharge to refresh your session. We use optional third-party analytics cookies to understand how you use GitHub.com so we can build better products. Find out more. We use optional third-party analytics cookies to understand how you use GitHub.com so we can build better products. You can always update your selection by clicking cookie preferences at the bottom of the page. For more information, see our privacy statement. We use essential cookies to perform essential functions of the website, for example, they are used to connect you. Learn more Always active We use analytical cookies to understand how you use our websites so that we can improve them, for example they are used to collect information about the pages you visit and how many clicks you need to accomplish a task. Learn more You can download a cheat sheet and install instructions for all the tools shown in this video. Transcript ctags is an external program that scans your code base, producing a keyword index. Vim has integrated ctags integration, which allows you to quickly navigate any code base that has been indexed by ctags. In the first half of this tutorial, we'll see what Vim can do when it's properly configured to work with ctags. In the second half, I'll show how to set up ctags to automatically index the code in your project, as well as all the bundled gems, and Ruby's standard library. Here I have the source code for evaluation, which is an open source project by thoughtbot. Let's open the lib/appraisal.rb file: vim lib/appraisal.rb Look at this: I'm going to position my cursor on the word Task, then use the control-close-bracket mapping to move to the definition of the current word. Boom! With a single command, we jumped to task class. This is a TaskLib rake class subclass. Let's try to invoke the jump-to-definition command on the superclass. Awesome! It's working, too! Note the path of this file: it is not part of the evaluation project, but I set up Vim to understand the project's dependencies. Let's try this one more time, good measure: we will move to the DSL definition. Well, it was three jumps, and it feels like we're a long way from where we started. The good news is that Vim maintains a history of jumps made using the command go to definition. We can inspect the battery by running the Command Ex tags: :tags - To mark from the line in the file / text 1 1 Task 4 lib / appraisal.rb 2 1 TaskLib 7 -/appraisal/lib/appraisal/task.rb 3 D SL 8 /usr/local/rbenv/versions/1.9.3-p327/lib/ruby/1.9.1/rake/tasklib.rb Most visited places appear last. You can think of the go-a-go. command definition as being like clicking on a hyperlink in a web browser. Vim also provides the equivalent of a rear button, which can be invoked by pressing in normal mode. Let's try that a few times, then inspect the list of tags again: ::tags to mark from the line in the file / text 1 1 Task 4 lib / appraisal.rb 'gt; 2 1 TaskLib 7 -/appraisal/lib/appraisal/task.rb 3 1 DS L 8 /usr/local/rbenv/versions/1.9.3-p327/lib/ruby/1.9.1/rake/tasklib.rb The greater than the sign indicates the current position in the label mat. If I use the back button a few more times, Vim ends up issuing the warning: at the bottom of the stack of tags Of course, we're back where we started. :Help tagstack Choose between multiple matches Let's see what happens when we invoke the command go to the definition on the evaluation. This brings us to the definition of a class called Evaluation, but look at this: the class is set in a module of the same name. So why did Vim choose to take us to the class definition instead of the module definition? The truth is that both definitions were indexed by ctags. When Vim is instructed to go to a tag for which there are multiple matches, he ranks the tags according to the priority rules, then jumps to the game with the highest priority. If you want to know more, you can read about the rules using Vim: :help tag-priority Let's go back to the mapping and try something different. Instead of using control closure support, I'll use the same prefixed command with the g key. This time, we get to see a list of all the tags that match the word Evaluation. pri kind tag file 1 F c Evaluation /home/vagrant/appraisal/lib/appraisal/appraisal.rb class:Appraisal class Evaluation 2 F m Appraisal /home/vagrant/appraisal/lib/appraisal/appraisal.rb module Evaluation 3 F m Evaluation/home/vagrant/appraisal/lib/appraisal/appraisal/command.rb module Evaluation 4 F m Evaluation/home/vagrant/appraisal/lib/appraisal/dependency.rb module Evaluation Each match is numbered, with various annotations, including the filepath. We can jump to one of these tags by entering their number and pressing the return. Let's try number 2 - and that turns out to be the module that contains a class of the same name. Vim provides some commands to help us interact with the list of matches. I can see the full list by running the ex:tselect command::tselect -pri type tag file 1 F c Evaluation /home/vagrant/appraisal/lib/appraisal/appraisal.rb class:Appraisal class Rating 'gt; 2 F m Rating /home/vagrant/appraisal/lib/appraisal/appraisal.rb module Evaluation 3 F m Evaluation /home/vagrant/appraisal/lib/appraisal/command.rb module Appraisal Note the greater than sign, which marks we have selected. After running:tselect, we can choose any item from the list by number, as we have done before. But we can also browse the list of matches using the tnext, tprev, tprev and tlast commands: Ex cmd Unimpaired mapping:tfirst [t:tnext]t:tlast]T Tim Pope's unimpaired plugin provides some handy to make it easier to execute these commands. I recommend installing it if you haven't already done so. Classes, modules and methods are all definitions So far I have shown that the command going to the definition can switch to module names and class names. He is also working on method definitions. Let's move on to task class and try this. I'm going to enter a template that allows me to jump to the method invocations: 'Try to jump to the definition of File.each: that works. What about appraisal.write_gemfile? It works, too. These methods are defined as part of the evaluation project, but here we use the fileutils module of Ruby's standard library. Of course, the command going to the definition works very well on the FileUtils module, and we can also use it to jump directly to the definition of the rm_f force remove method. Ex Vim commands has a couple of Ex commands that complete the go to the mapping definition: 'LT;C-]-tag 'keyword' - go to the first match for 'keyword' g-lt;C-] :tjump 'keyword' - invite the user to select from several matches for 'keyword'The command:tag accepts a keyword as an argument, and jumps directly to the first match (this behavior is similar to the close control support command). The command:tjump also goes directly to the first game if there is only one match, but if there are several matches, it displays a menu encouraging the user to choose where to go (as well as mapping). In addition,:tag and:tjump can accept a model! If the method you want to search for is right there in front of you, it's probably faster to position your cursor on the keyword and use one of the go to the mapping definition. But in many cases, you may want to look for the definition of a method you haven't used yet. This is where these Ex commands come into effect, especially because they cling to the completion behavior of Vim tabs. Suppose we work with the minitest library, and we'd like to know what assertions are available. Let's use the command:tag Ex to search for the definition of assert:tag assert Press enter, and boom! We go directly to the source of the method. We could scroll through the rest of this file to browse through the other claims, but here's another way. Enter the same Command Ex, but instead of pressing in, I'll use control dee: :tag affirming assert_block assert_block assert_equal assert_empty assert_headers_equal assert_equal assert_no_match assert_in_delta assert_not_equal assert_in_epsilon assert_not_nil assert_includes assert_not_same assert_instance_of assert_not_send assert_kind_of assert_nothing_raised assert_match assert_nil assert_path_exists assert_operator assert_performance assert_output assert_performance_constant assert_raises assert_performance_exponential assert_respond_to assert_performance_linear assert_same assert_performance_power assert_send assert_raise assert_silent assert_respond_to assert_throws assert_send Qui révèle toutes les balises correspondantes, nous donnant une liste concise</C-d> de tous les</C-]> </C-]> </C-]> </C-]> statements provided by minitest. Press the key cycles of the tab forward through the list. We've seen what Vim is capable of when configured correctly to work with ctags. Let's focus now on how to set up your own environment in the same way. First and foremost, we need to install exuberant ctags. Mac users: Beware that OS X ships with a BSD program that goes by the name of ctags. man -M /usr/share/man ctags This is not the program you want! You can get exuberant ctags via homebrew: brew install ctags Make sure that exuberant executable ctags appears on your way before the BSD program. who -a ctags /usr/local/bin/ctags /usr/bin/ctags /usr/local/bin/ctags If you set it up right, you should get a message like this when you run ctags dash-dash version: $ ctags --Exuberant version Ctags 5.8, Copyright (C) 1996-2009 Darren Hiebert Compiled: Feb 1 2013, 15:58:37 'dhiebert@users.sourceforge.net', Optional Compiled Features: 'wildcards, This virtual machine is running Ubuntu, and I can get the program running: sudo apt-get install exuberant-ctags basic use make Vim aware of the location of the tag file Here I have a copy of the evaluation project that has not yet been indexed. If I try to use vim go to the definition mapping on the word Rating, Vim reports an error: tag unse found Passons to the shell and invoke ctags to index this code base: ctags -R . The less big are flag tells the ctags to reappear through sub-repertory. Now there's a tag file in our project directory: ls -l Grep tags Out of curiosity, let's open this: vim tags Each line of the file represents a tag, which consists of a name, a file path and a search template that Vim can use to locate the label in the specified file. Also note that entries are sorted, allowing Vim to search for the index quickly. If we open our source code now, definition mapping must work for all modules, classes or methods that are defined in this project. Vim automatically searches for a file called tags in the current work directory, so in this case we didn't have to do any more configuration. But if we had saved the tags file to another location, then we could inform Vim where to search for it by setting the tag option: :Help 'tags' We can specify several tag files, and Vim will search each in turn until it finds a match. In short, two steps are required to make Vim's beacon feature work: the code base must be indexed to Vim ctags must be able to locate the index file of the tags. Instead of running them by hand, it would be ideal if we could configure our environment to manage them automatically. In the next sections, we'll look at how to make it work for your project, for your bundled gems, and finally for the standard ruby library. We credit Tim Pope for developing these strategies and for writing each of the required plugins. Installing Vim plugins For this demo, I use a Vim plugin manager called Vundle. dhiebert@users.sourceforge.dhiebert@users net start with a minimal setup, then we'll install each of these plugins and see how they help. I also created a custom command: TagFiles That inspects the Vim tags option, showing one entry per line. Tim Pope has devised a strategy that uses git hooks to automatically execute ctags on your project code base every time you check, commit, merge or rebase. The setup is a bit fiddly: you have to create a global git model containing some different shell scripts. But the beauty of this solution is that you only have to set it up once on your machine, then all your projects that are managed by git will be automatically indexed by ctags. I followed Tim Pope's instructions off camera. Now I can run: git init to apply the model to this repository, and git ctags to index the basis of evaluation code. Instead of generating a tag file in the root of the project, which would require you to add an extra line to your .gitignore file, Tim Pope recommends saving the tags file in the .git directory. Install the fugitive plugin: save the file's change source and run the beam installation The fugitive plugin will set up Vim to search for a label file in the .git directory. And now the command go to the definition will work for modules, classes and methods defined in this project. The tag file will become obsolete as we make changes to the code base. For example, if I rename the Task class to something else, then try to go to the new definition, Vim tells us that it can't find the label. Now, if I commit these changes: git add lib git commit -m's/Task/Quest' the post-commit crochet kicks in, re-indexing our code base with ctags. Now I can move on to the definition of the Quest class. The tag file will also be regenerated as a result of a case, a merger or a rebase, which means that your project is almost always indexed. Note that these git hooks will automatically enter for all repositories that you boot or clone after creating the model, but for git repositories that were created before the pattern, the hooks will not apply. You can solve this problem by running: git init in the project root of each of your pre-existing git repositories. We still can't use the definition card go on bundled gems: when I use it on the Rake module I get an error. Let's fix this. [gem-ctags] [] is a plugin for Rubygems that automatically summons ctags on gems as they are installed. The plugin itself is a rubygem, so you can install it running: gem install gem-ctags Now we Run: Gem ctags to index all rubygems that are already installed on the system. You won't have to run this command again though. From now on, every time you install a rubygem, it will be automatically indexed by ctags. Now all we have to do is make sure that Vim can find these index files tags, and we'll be able to use the vim ctags navigation features on the installed rubygems. The vim-bundler plugin tells Vim where to find the index file of tags for each gem listed in a project's gemfile. Let's allow the vim-bundler vim-bundler then restart Vim and inspect the list of tag files:: TagFiles Of course, it refers to each of the grouped gems. Now it works when I use the go to rak definition mapping. We still can't use the definition map going to mapping on Ruby's standard library. For example, nothing happens when I use it on the Force Removal method of the FileUtils module. If you use rbenv to manage your ruby environment, you need to install Tim Pope's rbenv-ctags plugin. I'll just copy and paste the instructions directly from the README... that's all there is to it. This plugin provides a handy command to run ctags to index the rubies that are managed by rbenv: rbenv ctags If you use ruby-build to install ruby versions, then this plugin will automatically index your rubies as they are installed. Now we just have to instruct Vim on where to find the tag index files for our standard ruby library. Recent versions of the vim-ruby plugin will take care of it. Even though this plugin comes with standard Vim distribution, I recommend installing it by hand to get the latest version. Now, if we inspect the tags option, it refers to the tag file in each directory from Vim's charging path: :TagFiles And of course, the command going to the definition now works on the methods set in Ruby's standard library. Outro To recap: we've set things up so that ctags automatically indexes our project, the bundled gems and Ruby's standard library. Thanks to fugitive plugins, bundler and ruby, Vim knows where to find the generated tag files. It takes a little effort to put it all together, but it is so helpful. You'll get a great boost from being able to use ctags in all your ruby projects. Projects.