

Cscope 的使用（领略 Vim + Cscope 的强大魅力）

1、Cscope 介绍

Cscope 是类似于 ctags 一样的工具，但可以认为她是 ctags 的增强版，因为她比 ctags 能够做更多的事。在 Vim 中，通过 cscope 的查询，跳转到指定的地方就像跳转到任何标签；她能够保存标签栈，所以通过合适的键盘映射绑定，你能够在函数向后或向前跳转，就像通常使用的 tags 一样。

首次使用 Cscope 时，他会根据源文件生成符号[数据库](#)。然后在以后的使用中，cscope 只是在源文件有改动或源文件列表不同时才会重建数据库。当在重建数据库时，未改动过的文件对应的数据库信息会从旧的数据库中拷贝过来，所以会使重建数据库快于一开始的新建数据库。

当你在命令行下调用 cscope 时，你会获得一个全屏选择窗口，能够使你查询特定的内容。然而，一旦你查询的有匹配，那么就会用你默认的编辑器来编辑该源文件，但是你不能简单的使用 Ctrl+]或者:tag 命令来从一个标签跳转到另一个标签。

Vim 中的 cscope 接口是通过以命令行形式调用完成的，然后解析查询返回的结果。最终的结果就是 cscope 查询结果就像通常的 tags 一样，这样你就可以自由跳转，就像在使用通常的 tags（用 ctrl+]或者:tag 跳转）。

2、Cscope 相关命令

所有的 cscope 命令都是通过向主 cscope 命令":cscope"传递参数选项。她最短的缩写是":cs"。":scscope"命令也做同样的事情并且同时会横向分隔窗口（简称："scs"）。

可用的缩写有：

add：增加一个新的 cscope 数据库/链接库

使用方法：

```
:cs add {file|dir} [pre-path] [flags]
```

其中：

[pre-path] 就是以 -p 选项传递给 cscope 的文件路径，是以相对路径表示的文件前加上的 path，这样你不要切换到你数据库文件所在的目录也可以使用它了。

[flags] 你想传递给 cscope 的额外旗标

实例：

```
:cscope add /root/code/vimtest/ftpd  
:cscope add /project/vim/cscope.out /usr/local/vim  
:cscope add cscope.out /usr/local/vim -C
```

find : 查询 cscope。所有的 cscope 查询选项都可用除了数字 5（“修改这个匹配模式”）。

使用方法：

```
:cs find {querytype} {name}
```

其中：

{querytype} 即相对应于实际的 cscope 行接口数字，同时也相对应于 nvi 命令：

- 0 或者 s —— 查找这个 C 符号
- 1 或者 g —— 查找这个定义
- 2 或者 d —— 查找被这个函数调用的函数（们）
- 3 或者 c —— 查找调用这个函数的函数（们）
- 4 或者 t —— 查找这个字符串
- 6 或者 e —— 查找这个 egrep 匹配模式
- 7 或者 f —— 查找这个文件
- 8 或者 i —— 查找#include 这个文件的文件（们）

实例：（#号后为注释）

```
:cscope find c ftpd_send_resp      # 查找所有调用这个函数的函数（们）  
:cscope find 3 ftpd_send_resp      # 和上面结果一样
```

:cscope find 0 FTPD_CHECK_LOGIN # 查找 FTPD_CHECK_LOGIN 这个符

号

执行结果如下：

```
Cscope tag: FTPD_CHECK_LOGIN
#   line  filename / context / line
1    19   ftpd.h <<GLOBAL>>
      #define FTPD_CHECK_LOGIN() /
2   648   ftpd.c <<ftpd_do_pwd>>
      FTPD_CHECK_LOGIN();
3   661   ftpd.c <<ftpd_do_cwd>>
      FTPD_CHECK_LOGIN();
4   799   ftpd.c <<ftpd_do_list>>
      FTPD_CHECK_LOGIN();
5   856   ftpd.c <<ftpd_do_nlst>>
      FTPD_CHECK_LOGIN();
6   931   ftpd.c <<ftpd_do_syst>>
      FTPD_CHECK_LOGIN();
7   943   ftpd.c <<ftpd_do_size>>
      FTPD_CHECK_LOGIN();
8   960   ftpd.c <<ftpd_do_dele>>
      FTPD_CHECK_LOGIN();
9   981   ftpd.c <<ftpd_do_pasv>>
      FTPD_CHECK_LOGIN();
```

Enter nr of choice (<CR> to abort):

然后输入最前面的序列号即可。

help : 显示一个简短的摘要。

使用方法：

:cs help

kill : 杀掉一个 cscope 链接（或者杀掉所有的 cscope 链接）

使用方法：

:cs kill {num|partial_name}

为了杀掉一个 cscope 链接，那么链接数字或者一个部分名称必须被指定。部分名称可以简单的是 cscope 数据库文件路径的一部分。要特别小心使用部分路径杀死一个 cscope 链接。

假如指定的链接数字为-1，那么所有的 `cscope` 链接都会被杀掉。

reset: 重新初始化所有的 `cscope` 链接。

使用方法:

`:cs reset`

show: 显示 `cscope` 的链接

使用方法:

`:cs show`

假如你在使用 `cscope` 的同时也使用 `ctags`，`|:cstag|` 可以允许你在跳转之前指定从一个或另一个中查找。例如，你可以选择首先从 `cscope` 数据库中查找，然后再查找你的 `tags` 文件（由 `ctags` 生成）。上述执行的顺序取决于 `|:csto|` 的值。

`|:cstag|` 当从 `cscope` 数据库中查找标识符时等同于 “`:cs find g`”。

`|:cstag|` 当从你的 `tags` 文件中查找标识符时等同于 “`|:tjump|`”。

3、Cscope 选项

使用 `|:set|` 命令来设置 `cscope` 的所有选项。理想情况是，你可以在你的启动文件中做这件事情（例如：`.vimrc`）。有些 `cscope` 相关变量只有在 `|.vimrc|` 中才是合法的。在 `vim` 已经启动之后再设置它们没有任何作用！

‘`cscopeprg`’指定了执行 `cscope` 的命令。默认是“`cscope`”。例如：

`:set csprg=/usr/local/bin/cscope`

‘`cscopequickfix`’指定了是否使用 `quickfix` 窗口来显示 `cscope` 的结果。这是一组用逗号分隔的值。每项都包含于 `|:csope-find|` 命令（`s, g, d, c, t, e, f`, 或者 `i`）和旗标（`+`, `-` 或者 `0`）。

‘`+`’预示着显示结果必须追加到 `quickfix` 窗口。

‘-’隐含着清空先前的显示结果，‘0’或者不设置表示不使用 **quickfix** 窗口。查找会从开始直到第一条命令出现。默认的值是’’（不使用 **quickfix** 窗口）。下面的值似乎会很有用：’’s-,c-,d-,i-,t-,e-’’。

假如‘**cscopetag**’被设置，然后诸如’:tag’和 **ctrl+]和**’vim -t’等命令会始终使用|:cstag|而不是默认的:tag 行为。通过设置‘cst’，你将始终同时查找 **cscope** 数据库和 **tags** 文件。默认情况是关闭的，例如：

```
:set cst
:set nocst
```

‘**csto**’

‘csto’的值决定了|:cstag|执行查找的顺序。假如‘csto’被设置为 0，那么 **cscope** 数据将会被优先查找，假如 **cscope** 没有返回匹配项，然后才会查找 **tag** 文件。反之，则查找顺序相反。默认值是 0，例如：

```
:set cst=0
:set cst=1
```

假如‘**cscopeverbose**’没有被设置（默认情况是如此），那么当在增加一个 **cscope** 数据库时不会显示表示执行成功或失败的信息。理想情况是，在增加 **cscope** 数据库之前，你应该在你的|.vimrc|中重置此选项，在增加完之后，设置它。此后，当你在 **vim** 中增加更多的数据库时，你会得到（希望是有用的）信息展示数据库增加失败。例如：

```
:set csverb
:set nocverb
```

‘**cspc**’的值决定了一个文件的路径的多少部分被显示。默认值是 0，所以整个路径都会被显示。值为 1 的话，那么就只会显示文件名，不带路径。其他值就会显示不同的部分。例如：

```
:set cspc=3
```

将会显示文件路径的最后 3 个部分，包含这个文件名本身。

4、在 **Vim** 中怎么使用 **cscope**

你需要做的第一步就是为你的源文件建立一个 `cscope` 数据库。大多数情况下，可以简单的使用”`cscope -b`”。

假设你已经有了一个 `cscope` 数据库，你需要将这个数据库 “增加” 进 `Vim`。那将会建立一个 `cscope` “链接” 并且使它能够被 `Vim` 所使用。你可以在你的 `.vimrc` 文件中做这件事，或者在 `Vim` 启动之后手动地做。例如，为了增加数据库”`cscope.out`”，你可以这样做：

```
:cs add cscope.out
```

你可以通过执行”`:cs show`”来再次检查以上执行的结果。这将会产生如下的输出：

#	pid	database name	prepend path
0	11453	cscope.out	<none>

提示：

由于微软的 RTL 限制，Win32 版本会显示 0 而不是真正的 pid。

一旦一个 `cscope` 链接建立之后，你可以查询 `cscope` 并且结果会反馈给你。通过命令”`:cs find`”来进行查找。例如：

```
:cs find g FTPD_CHECK_LOGIN
```

执行以上命令可能会变得有点笨重的，因为它要做相当的输入次数。假如有不止一个匹配项，你将会被提供一个选择屏幕来选择你想匹配的项。在你跳转到新位置之后，可以简单的按下 `ctrl+t` 就会返回到以前的一个。

5、建议的用法

将如下内容放置到你的 `.vimrc` 中：

```
if has("cscope")
    set csprg=/usr/local/bin/cscope
    set csto=0
    set cst
    set nocsverb
    " add any database in current directory
    if filereadable("cscope.out")
        cs add cscope.out
    " else add database pointed to by environment
    elseif $CSCOPE_DB != ""
```

```

        cs add $CSCOPE_DB
    endif
    set csverb
endif

```

通过设置'cscopetag'，我们已经有效的将所有:tag 的情况都替换为:cstag。这包括:tag、ctrl+]，和"vim -t"。然后，正常的 tag 命令就会不光在 tag 文件中查找，也会在 cscope 数据库中查找。

有些用户可能想保留常规的 tag 行为并且有一个不同的快捷方式来使用:cstag。例如，可以使用如下命令来映射 ctrl+_（下划线）到:cstag：

```
map <C-_-> : cstag <C-R>=expand("<word>")<CR><CR>
```

一些经常用 cscope 查找（使用":cs find"）是查找调用某一特定函数的所有函数，和查找所有出现特定 C 符号的地方。为了做这些事，你可以使用如下的键盘映射作为例子：

```
map g<C-]> :cs find 3 <C-R>=expand("<word>")<CR><CR>
map g<C-/> :cs find 0 <C-R>=expand("<word>")<CR><CR>
```

这些给 ctrl+]（右中括号）和 ctrl+/（反斜杠）的映射可以允许你将光标放置到函数名称或者 C 符号上然后执行快速 cscope 查找匹配。

或者你可以使用如下方案（很好用，可以将其添加到.vimrc 中）：

```

nmap <C-_->s :cs find s <C-R>=expand("<word>")<CR><CR>
nmap <C-_->g :cs find g <C-R>=expand("<word>")<CR><CR>
nmap <C-_->c :cs find c <C-R>=expand("<word>")<CR><CR>
nmap <C-_->t :cs find t <C-R>=expand("<word>")<CR><CR>
nmap <C-_->e :cs find e <C-R>=expand("<word>")<CR><CR>
nmap <C-_->f :cs find f <C-R>=expand("<file>")<CR><CR>
nmap <C-_->i :cs find i <C-R>=expand("<file>")<CR><CR>
nmap <C-_->d :cs find d <C-R>=expand("<word>")<CR><CR>

```

“ 使用'ctrl - 空格'，然后查找时就会使 vim 水平分隔窗口，结果显示在

“ 新的窗口中

```

nmap <C-Space>s :scs find s <C-R>=expand("<word>")<CR><CR>
nmap <C-Space>g :scs find g <C-R>=expand("<word>")<CR><CR>
nmap <C-Space>c :scs find c <C-R>=expand("<word>")<CR><CR>
nmap <C-Space>t :scs find t <C-R>=expand("<word>")<CR><CR>
nmap <C-Space>e :scs find e <C-R>=expand("<word>")<CR><CR>

```

```
nmap <C-Space>f :scs find f <C-R>=expand("<cfiler>")<CR><CR>
nmap <C-Space>i :scs find i <C-R>=expand("<cfiler>")<CR><CR>
nmap <C-Space>d :scs find d <C-R>=expand("<cword>")<CR><CR>
```

“ 两次按下’ ctrl – 空格’, 然后查找时就会竖直分隔窗口而不是水平分隔

```
nmap <C-Space><C-Space>s
/:vert scs find s <C-R>=expand("<cword>")<CR><CR>
nmap <C-Space><C-Space>g
/:vert scs find g <C-R>=expand("<cword>")<CR><CR>
nmap <C-Space><C-Space>c
/:vert scs find c <C-R>=expand("<cword>")<CR><CR>
nmap <C-Space><C-Space>t
/:vert scs find t <C-R>=expand("<cword>")<CR><CR>
nmap <C-Space><C-Space>e
/:vert scs find e <C-R>=expand("<cword>")<CR><CR>
nmap <C-Space><C-Space>i
/:vert scs find i <C-R>=expand("<cfiler>")<CR><CR>
nmap <C-Space><C-Space>d
/:vert scs find d <C-R>=expand("<cword>")<CR><CR>
```

6、结合实际来使用 cscope

我这里有一个 ftp 服务器的工程，主要文件如下（Secure CRT vt100, traditional, 13）：

```
[root@localhost ftpd]# ls
dxyh.h          dxyh_thread_lib.c  ftpd.c          Makefile
dxyh_lib.c      error.c            ftpd.h          record.c
dxyh_thread.h  error.h            ftpd_main.c     record.h
[root@localhost ftpd]#
```

下面就是要 cscope 命令来建立数据库文件（多了 3 个和 cscope 相关的文件）：

```
[root@localhost ftpd]# cscope -Rbq *
[root@localhost ftpd]# ls
cscope.in.out  dxyh_lib.c          error.h          Makefile
cscope.out     dxyh_thread.h       ftpd.c           record.c
cscope.po.out  dxyh_thread_lib.c  ftpd.h           record.h
dxyh.h         error.c              ftpd_main.c
[root@localhost ftpd]#
```

说明：

a、 cscope 的选项分析：

- R ： 表示包含此目录的子目录，而非仅仅是当前目录；
- b ： 此参数告诉 cscope 生成数据库后就自动退出；

-q : 生成 `cscope.in.out` 和 `cscope.po.out` 文件, 加快 `cscope` 的索引速度

可能会用到的其他选项:

-k : 在生成索引时, 不搜索 `/usr/include` 目录;

-i : 如果保存文件列表的文件名不是 `cscope.files` 时, 需要加此选项告诉 `cscope` 到哪里去找源文件列表;

-I dir : 在 **-I** 选项指出的目录中查找头文件

-u : 扫描所有文件, 重新生成交叉索引文件;

-C : 在搜索时忽略大小写;

-P path: 在以相对路径表示的文件前加上的 **path**, 这样你不用切换到你数据库文件的目录也可以使用它了。

说明: 要在 **VIM** 中使用 `cscope` 的功能, 需要在编译 **Vim** 时选择 `”+cscope”`。**Vim** 的 `cscope` 接口会先调用 `cscope` 的命令行接口, 然后分析其输出结果找到匹配处显示给用户。

b、 若是不指定 **-b** 选项, 则在建立完数据库后进入如下界面:

```

C symbol: FTPD_CHECK_LOGIN

  File      Function      Line
0 ftpd.h <global>          19 #define FTPD_CHECK_LOGIN() \
1 ftpd.c ftpd_do_pwd      648 FTPD_CHECK_LOGIN();
2 ftpd.c ftpd_do_cwd      661 FTPD_CHECK_LOGIN();
3 ftpd.c ftpd_do_list     799 FTPD_CHECK_LOGIN();
4 ftpd.c ftpd_do_nlst     856 FTPD_CHECK_LOGIN();
5 ftpd.c ftpd_do_syst     931 FTPD_CHECK_LOGIN();
6 ftpd.c ftpd_do_size     943 FTPD_CHECK_LOGIN();
7 ftpd.c ftpd_do_dele     960 FTPD_CHECK_LOGIN();
8 ftpd.c ftpd_do_pasv     981 FTPD_CHECK_LOGIN();
9 ftpd.c ftpd_do_port    1039 FTPD_CHECK_LOGIN();
a ftpd.c ftpd_do_mkd     1078 FTPD_CHECK_LOGIN();
b ftpd.c ftpd_do_rmd     1106 FTPD_CHECK_LOGIN();
c ftpd.c ftpd_do_type    1137 FTPD_CHECK_LOGIN();

* 3 more lines - press the space bar to display more *
Find this C symbol: FTPD_CHECK_LOGIN
Find this global 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:
Find all function definitions:
Find all symbol assignments:

```

这里是想要查找 C 符号：FTPD_CHECK_LOGIN，你可以通过按 Tab 键来进行匹配内容和输入项的切换。按下 ctrl+d 退出。

注意：在此时，不可以使用 ctrl+] 进行跳转！

下面用 Vim 打开其中的一个文件进行编辑，然后看看使用 cscope 的具体例子：

输入：vim ftpd.c

```

#include <netinet/in.h>
#include <arpa/inet.h>
#include <getopt.h>
#include "dxyh.h"
#include "dxyh_thread.h"
#include "ftpd.h"

#include "error.h"
#include "record.h"

extern log_t *logfd;

static void ftpd_usage(void);
static void ftpd_verbose(void);
static void ftpd_help(void);
static void ftpd_sig_chld(int signo);
static void ftpd_sig_int(int signo);

```

看到此时光标在 `ftpd_help` 这个函数声明上，现在若我们想要看看这个函数是怎么实现的，可以有如下方法：

- 1) 直接按下 `ctrl+]` # 就是按下 `ctrl` 键的同时按下 `']'` 键
- 2) 按下 `ctrl+_g` # 按下 `ctrl` 键和下划线（同时按下 `shift` 和 `'_'` 键）和 `g`
- 3) 输入 “`:cs find g ftpd_help`” 后回车
- 4) 输入 “`:tag ftpd_help`” # 假如有安装 `ctag` 的话

然后就会进行跳转：

```
/*
 * show help
 */
static void ftpd_help(void)
{
    printf("You can use any ftp client software to login.\n"
           "Cmds as follows are supported:\n"
           "user ls cd pwd get mget delete put mput\n"
           "size port passive ascii binary dir mkdir\n"
           ".....\n"
           "Well, you can add more cmds!\n");
}
```

小结：在非 windows 系统上很多人都会选择强大的 Vim 作为编辑器，同时，我们要是能够用好那些同样强大的插件的话，那提高的战斗力的可不止一点哦。常常会听到类似的抱怨，linux 下没有好用的 IDE，殊不知，用 Vim 结合一些插件，同样可以拥有 IDE 的强大功能，看代码也不错，可以有类似 source insight 的功能。这里展示下我的 Vim，可能有些简陋，但至少有了些 IDE 的影子了：

```
[1:ftpd.c]*[6:ftpd_main.c]
-MiniBufExplorer- 1,1 All
"= /root/code/vimtest/ftpd/
../
Makefile
cscope.in.out
cscope.out
cscope.po.out
[File List] 4,1 20%
| ftpd_quit_flag
| ftpd_hash_print
| ftpd_tick_print
| ftpd_serv_port
| ftpd_cur_dir
| ftpd_cur_pasv_fd
| ftpd_cur_pasv_connfd
| ftpd_cur_port_fd
| ftpd_cur_type
| ftpd_cur_user
| ftpd_nchild
| pids
| ftpd_cmds
| ftpd_users
| ftpd_serv_resps
- function
| ftpd_debug
| ftpd_init
| ftpd_parse_args
Tag_List__ 10,5 12% ftpd.c 78,15-18 4%

static int ftpd_get_list_stuff(char buff[], size_t len);
static int get_file_info(const char *filename,
    char buff[], size_t len);
static int dir_path_ok(const char *dir_path);
static void ftpd_close_all_fds(void);
static void parent_atlast(void);
static void pr_cpu_time(void);
static void ptransfer(const char *direction, long bytes,
    const struct timeval *t0,
    const struct timeval *t1);
static int add2pids(pid_t pid);
static void dele_from_pids(pid_t pid);

int ftpd_debug_on;
int ftpd_record_on;
int ftpd_quit_flag;
int ftpd_hash_print;
int ftpd_tick_print;
uint16_t ftpd_serv_port;
char ftpd_cur_dir[PATH_MAX];
int ftpd_cur_pasv_fd;
int ftpd_cur_pasv_connfd;
int ftpd_cur_port_fd;
int ftpd_cur_type;
const struct ftpd_user_st *ftpd_cur_user;
int ftpd_nchild;
pid_t pids[MAX_CHIHLD_NUM];
```

对了，还有一点：默认情况下 cscope 值会在当前目录下针对 c、iex 和 yacc（扩展名分别为.c、.h、.I、.y）程序文件进行解析（如果指定了-R 参数则包含其自身的子目录）。这样出现的问题就是，我们对于 C++或 [Java](#) 文件怎么办，解决方案是：我们可以生成一个名为 cscope.finds 的文件列表，并交由 cscope 去解析。在 Linux 系统中，生成这个文件列表的方法是：

```
find . -name "*.java" > cscope.files
```

然后运行 cscope -b 命令重新生成数据库就 OK 了。

好了，这里就先介绍这么多吧，更多用法请查阅相关资料。有空我把 Vim 的使用再说下。有问题希望大家不吝赐教，欢迎交流。