

拼装的艺术：vim 之 IDE 进化实录

```
// =====  
// 版本信息：0.1  
// 作者姓名：杨新涛  
// 电子邮件：yangyang.gnu@gmail.com  
// 博客地址：http://hi.csdn.net/yangyang\_gnu  
// 更新时间：2011 年 7 月 28 日 21 点 25 分  
// 版权信息：本文版权归杨新涛所有。非商业转载，请保留文档信息；商业转载，须经得本人同意  
// =====
```

Ken Thompson 告诉我们——“一个程序只实现一个功能，且做到极致，多个程序协作实现复杂任务”——这是 unix。是滴，这种哲学在 linux 上随处可见，比如，vim 与她的插件们（怎么感觉像在说白雪公主与七个小矮人 -_- \$ ）。下面开始我们的 vim 之 IDE 进化之旅吧。

这个时代，上规模的软件项目已不可能用简单的文本编辑器完成，IDE 是必然选择。linux 下 IDE 大致分为两类：“品牌机”和“组装机”。“品牌机”中有些（开源）产品还不错，比如：codeblocks、netbeans、eclipse、anjuta 等等，对于初涉 linux 开发的朋友而言是个不错的选择（我指的是 codeblocks），但对于老鸟来说总有这样那样的欠缺。听闻 linux torvalds 这类大牛用的是类 emacs（准确的说是 microemacs）和一堆插件拼装而成的 IDE，为向大牛致敬，加之我那颗“喜欢折腾”的心，我也选择“组装机”方式。首要任务，选择编辑器。

linux 上存在两种编辑器：神之编辑器——emacs，编辑器之神——vim。关于 emacs 与 vim 孰轻谁重之争已是世纪话题，我无意参与其中，在我眼里，二者都是创世纪的优秀编辑器，至少在这个领域作到了极致，它们让世人重新认识了编辑操作的本质——用命令而非键盘——去完成编辑任务。好了，如果你不是 emacs 控，不要浪费时间再去比较，选择学习曲线相较平滑的那个直接啃 man 吧——vim 不会让你失望的。

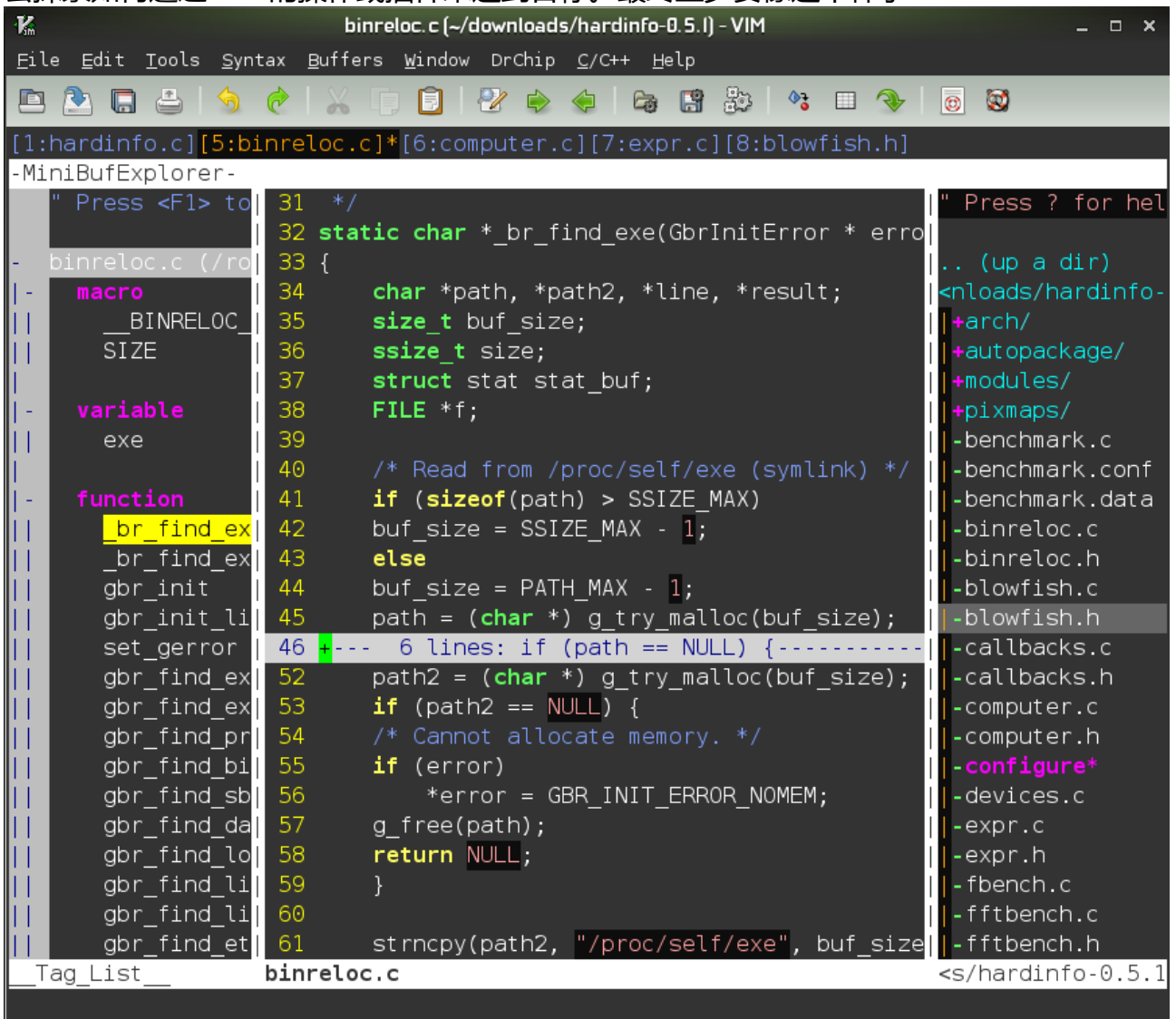
对于 vim 的喜爱，我无法用言语表述，献上一首 vi 的湿哥哥（-_-#）以表景仰之情：

我心之禅如同 vi 之大道，
vi 之漫路即为禅修，
vi 之命令如禅之心印，
未得此道者视之怪诞，
与之相伴者洞其真谛，
长修此道者巨变人生。

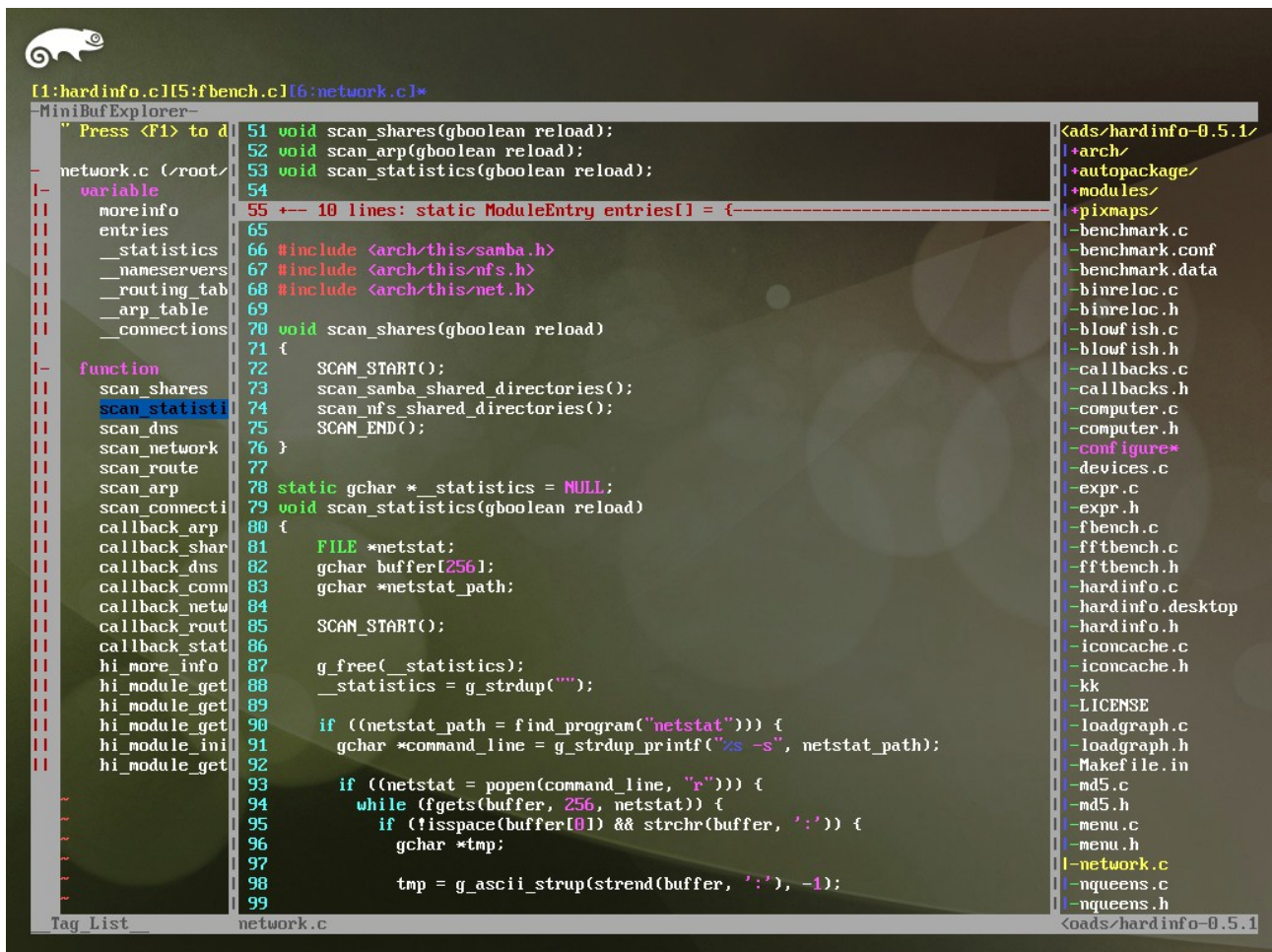
——作者：reddy@lion.austin.com，
译作：yangyang.gnu@gmail.com

OK，言归正传，说说 vim 用于代码编写提供了哪些直接和间接功能支撑。vim 联机手册中，50%的例子都是在讲 vim 如何高效编写代码，由此可见，vim 是一款面向程序员的编辑器，即使某些功能 vim 无法直接完成，借助其丰富的插件资源，必定可以达成目标（注，推荐两份 vim 入门资料：《vim 用户手册中文版 7.2》、《A Byte of Vim v0.51 (for Vim version 7)》）。

我是个“目标驱动”的信奉者，本文内容，我会先给出优秀 IDE 应具备哪些功能，再去探索如何通过 vim 的操作或插件来达到目标。最终至少要像这个样子：



(图 0：图形环境下 IDE 总揽)



(图 1：纯字符模式下 IDE 总揽)

在介绍功能 IDE 应具备的功能之前，先说说 vim 插件相关事宜。vim 有一套自己的脚本语言，通过这种脚本语言可以实现与 vim 交互的，达到功能扩展的目的。一组 vim 脚本就是一个 vim 插件，vim 的很多功能都是通过其插件实现，在官网上有丰富的插件资源，任何你想得到的功能，如果 vim 无法直接支持，那一般都有对应的插件为你服务，有需求时可以去逛逛。

vim 插件目前分为两类：*.vim 和*.vba。前者是传统格式的插件，实际上就是一个文本文件，通常 *someplugin.vim*（插件脚本）与 *someplugin.txt*（插件帮助文件）并存在一个打包文件中，解包后将 *someplugin.vim* 拷贝到 *~/vim/plugin/* 目录，*someplugin.txt* 拷贝到 *~/vim/doc/* 目录即可完成安装，重启 vim 后刚安装的插件就已经生效，但帮助文件需执行 *helptags ~/vim/doc/* 才能生效。传统格式插件需要解包和两次拷贝才能完成安装，相对较繁琐，所以后来又出现了*.vba 格式插件，安装更便捷，只需在 shell 中依次执行如下命令即可：

```
vim someplugin.vba
:so %
:q
```

另外，后面涉及到的各类插件，只介绍了我常用的操作，有时间，建议看看它们的帮组文档 (:h *someplugin*) 。

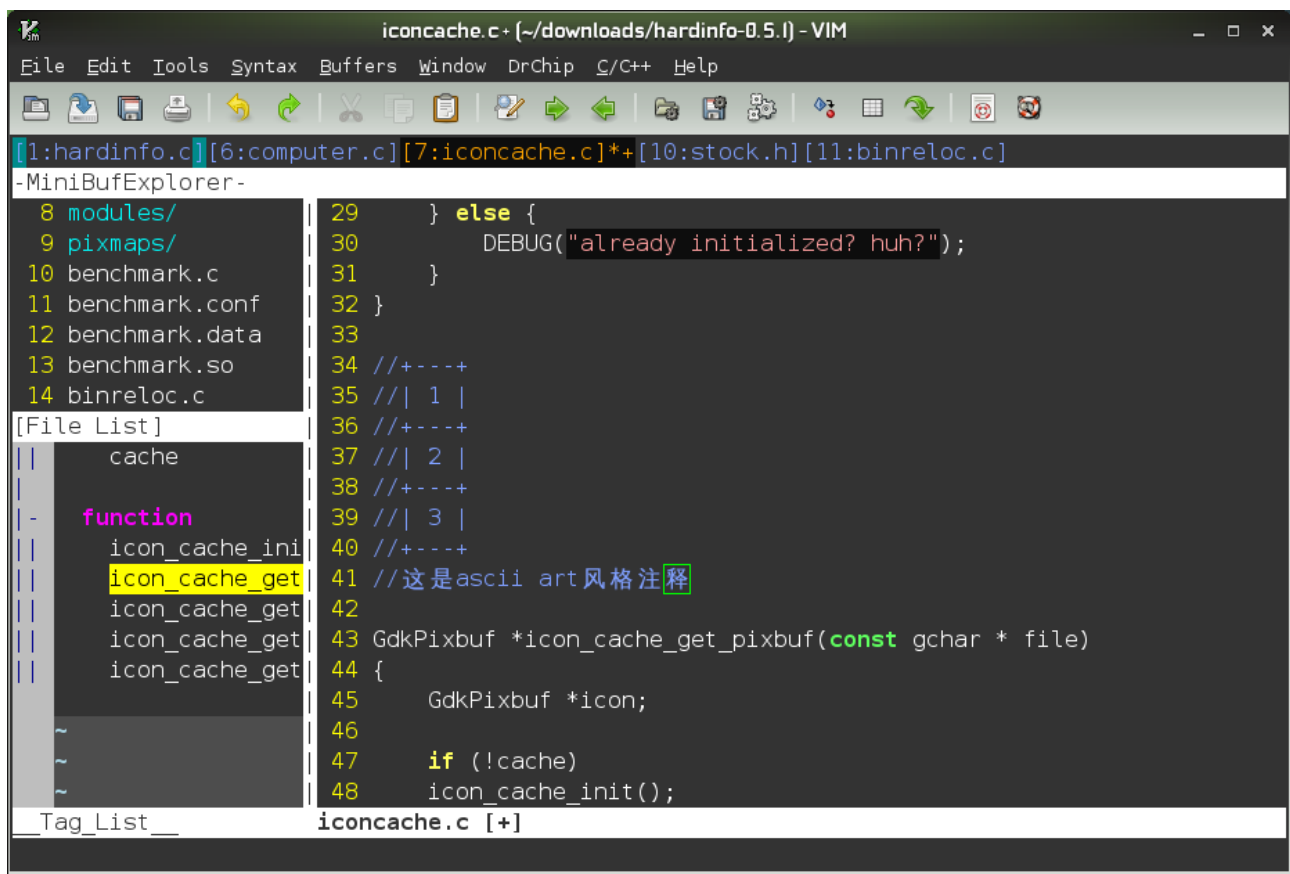
[- 注释与反注释 -]

注释时到每行代码前输入//，取消注释时再删除//，这种方式不是现代人的行为。IDE 应该支持对选中文本块批量（每行）添加注释符号，反之，可批量取消。本来 vim 通过宏方式可以支持该功能，但每次注释时要自己录制宏，关闭 vim 后宏无法保存，所以有人专门编写了一款插件，其中部分功能就是快速注释与反注释。

- 插件名：c-support.vim
- 常用操作：
 - \cc，用 CPP 语法风格注释掉选中文本块或当前行
 - \co，取消选中文本块或当前行的 CPP 语法风格注释
- 注意：由于 C 风格注释有“嵌套注释”风险，我都用 CPP 风格注释。

此外，有时需要 ascii art 风格注释，推荐如下插件：

- 插件名：DrawIt.vim
- 常用操作：
 - :Dstart，开始绘制结构化字符图形
 - :Distop，停止绘制结构化字符图形
 - 空格，绘制或擦出字符



(图 2 : ascii art 风格注释)

[- 全能补全 -]

提升编码效率的王牌功能就是智能补全。试想下，有个函数叫 `get_count_and_size_from_remotefile()`，当你输入 “`get_`” 后 IDE 自动帮你输入完整的函数名，又有个文件 `~yangyang.gnu/this/is/a/deep/dir/myfile.txt`，就像在 shell 中一样，类似 `tab` 键的东东自动补全文件路径那是何等的惬意啊！以上两个例子仅是我需要的补全功能的一部分，完整的补全应具备：1) 预处理语句、语法语句、函数框架补全；2) 函数名、变量名、结构名、结构成员、头文件名、文件路径。

对于第一类补全，也可以借助 `c-support.vim` 实现。严格地说，这不叫智能补全，仅是 `c-support.vim` 是对预处理语句、语法语句、函数框架等设定了快捷键而已，如，要写 `do-while` 语句只需简单的输入 “`\sd`”，要包括头文件输入 “`\p<`” 即可出现 `#include <XX>`。

- 插件名：`c-support.vim`

- 操作：

`\p<` `#include <XX>`

`\p"` `#include "XX"`

`\pd` `#define`

```
\sd  do{}-while
\sfo  for(){}
\sif  if(){}
\sife if(){}else{}
\se   else{}
\swh  while(){}
\ss   switch(){}
\cfu  函数框架
```

- 注意：所有模板位于\$HOME/.vim/c-support/templates/目录，可按个人偏好更新。

第二类补全就真的智能了。实现智能补全的原理非常简单：代码中所有函数、结构、成员、变量、宏等对象的名字、所在文件路径、定义、类型等信息（称之为标签信息）保存到一个独立文件（称之为数据库文件）中，vim 和智能补全插件根据数据库信息快速匹配输入的字符，若找到匹配的则以列表形式显示之。

- 软件：ctags
- 插件：new-omni-completion（内置）
- 操作：A、生成标签数据库文件。在你项目所在目录的顶层执行 ctags -R，该目录下会多出个 tags 文件；

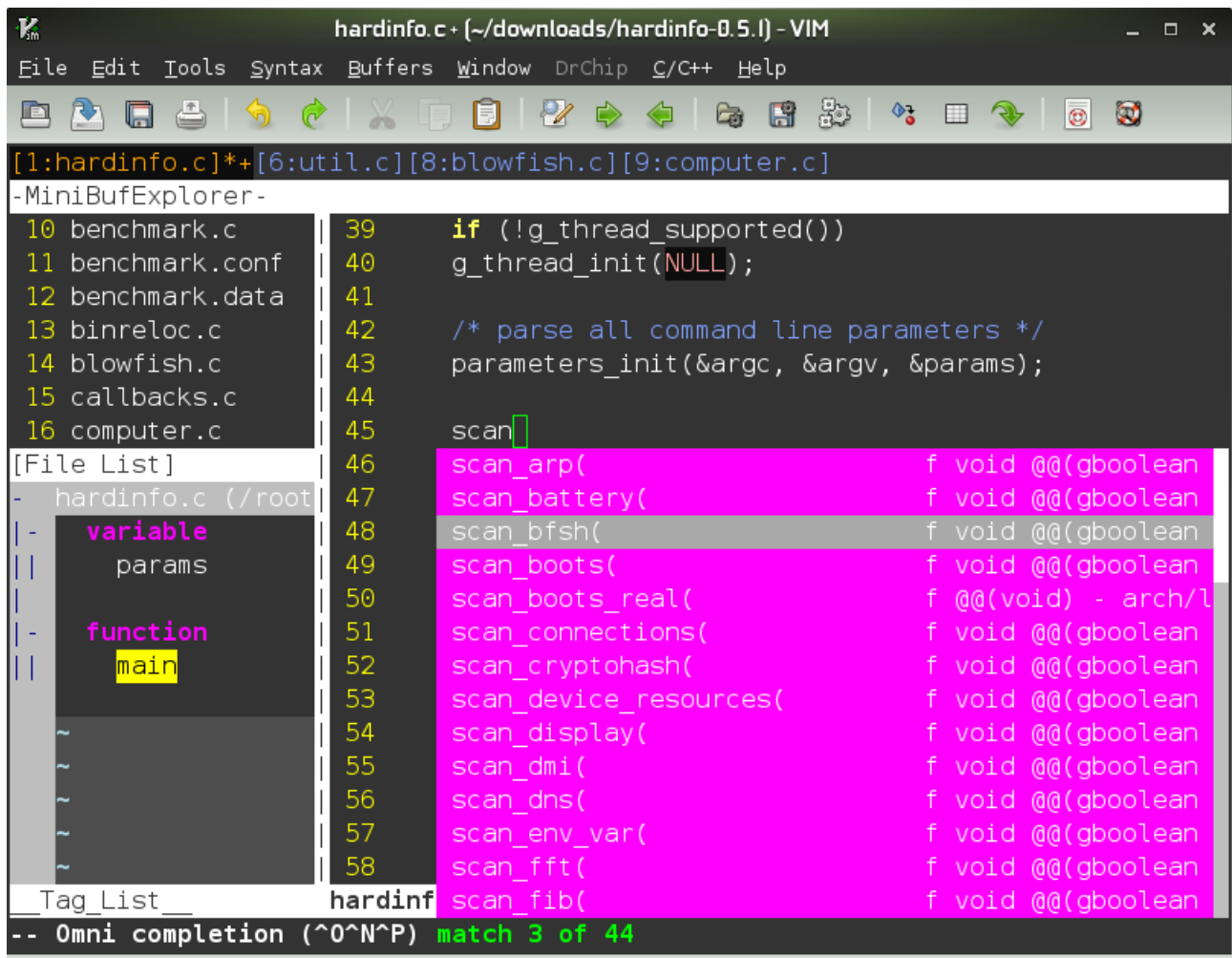
B、在 vim 中引入标签数据库文件。在 vim 中执行命令:set tags=/home/your_proj/tags

C、在.vimrc 中增加如下配置信息：

```
"开启文件类型侦测
filetype on
"根据侦测到的不同类型加载对应的插件
filetype plugin on
"根据侦测到的不同类型采用不同的缩进格式
filetype indent on
"取消补全内容以分割子窗口形式出现，只显示补全列表
set completeopt=longest,menu
```

D、需要进行函数名、变量名、结构名、结构成员补全时输入 Ctrl+X Ctrl+O，需要头文件名补全时输入 Ctrl+X Ctrl+I，需要文件路径补全时输入 Ctrl+X Ctrl+F。

具体效果如以下几图所示：



```
hardinfo.c · (~/downloads/hardinfo-0.5.1) - VIM
File Edit Tools Syntax Buffers Window DrChip C/C++ Help

[1:hardinfo.c]*+[6:util.c][8:blowfish.c][9:computer.c]
-MiniBufExplorer-
10 benchmark.c
11 benchmark.conf
12 benchmark.data
13 binreloc.c
14 blowfish.c
15 callbacks.c
16 computer.c
[File List]
- hardinfo.c (/root
| - variable
| | params
| - function
| | main
| ~
| ~
| ~
| ~
__Tag_List__ hardinf
-- Omni completion (^0^N^P) match 3 of 44

39 if (!g_thread_supported())
40 g_thread_init(NULL);
41
42 /* parse all command line parameters */
43 parameters_init(&argc, &argv, &params);
44
45 scan
46 scan_arp( f void @@(gboolean
47 scan_battery( f void @@(gboolean
48 scan_bfsh( f void @@(gboolean
49 scan_boots( f void @@(gboolean
50 scan_boots_real( f @@(void) - arch/l
51 scan_connections( f void @@(gboolean
52 scan_cryptohash( f void @@(gboolean
53 scan_device_resources( f void @@(gboolean
54 scan_display( f void @@(gboolean
55 scan_dmi( f void @@(gboolean
56 scan_dns( f void @@(gboolean
57 scan_env_var( f void @@(gboolean
58 scan_fft( f void @@(gboolean
scan_fib( f void @@(gboolean
```

(图 3 : 函数名补全)

```
hardinfo.c+ (~/.downloads/hardinfo-0.5.l) - VIM
File Edit Tools Syntax Buffers Window DrChip C/C++ Help

[1:hardinfo.c]*+[6:util.c][8:blowfi
ger.c]
-MiniBufExplorer-
32 report.c
33 sha1.c
34 shell.c
35 socket.c
36 stock.c
37 syncmanager.c
38 tags
[File List]
|
|- function
|| main
~
~
~
~
~
~
~
Tag_List__ hardinfo.c [+]
-- Omni completion (^0^N^P) match 2 of 20

33 {
34     GSList
35
36     DEBUG(
37
38     DEBUG(
39     if (!g
40     g_thre
41
42     /* par
43     parame
44
45     params.
46     /* show version information and quit */
47     if (params.show_version) {
48     g_print("HardInfo version " VERSION "\n");
49     g_print
50         ("Copyright (C) 2003-2009 Leandro A. F. Pereira
51

argv0 m gchar
autoload_deps m gboolean
create_report m gboolean
gui_running m gboolean
list_modules m gboolean
path_data m gchar
path_lib m gchar
report_format m gint
show_version m gboolean
use_modules m gchar
_ProgramParameters::argv0 m gchar
_ProgramParameters::autoload_deps m gboolean
_ProgramParameters::create_report m gboolean
_ProgramParameters::gui_running m gboolean
_ProgramParameters::list_modules m gboolean
```

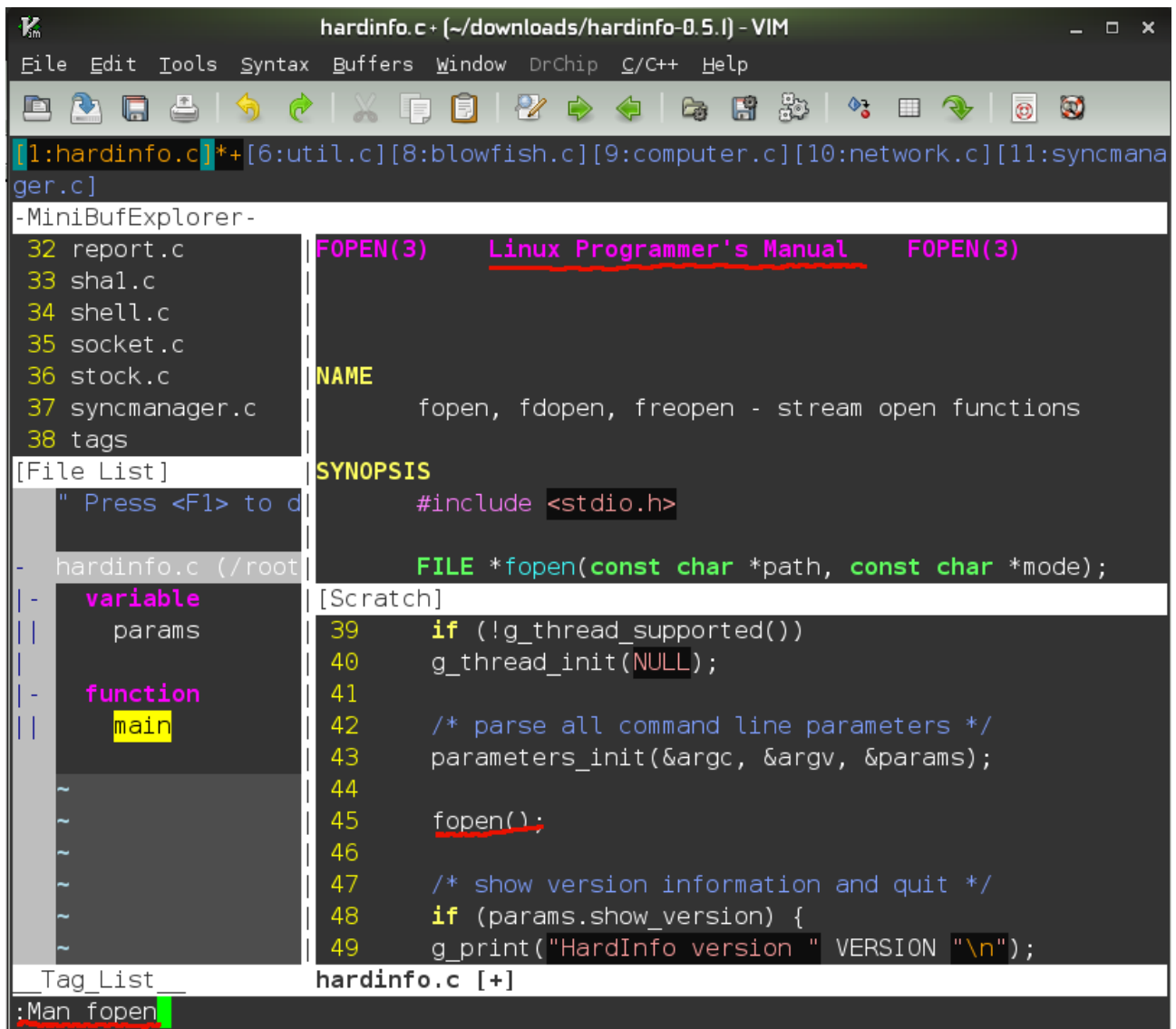
(图 4 : 结构成员补全)

- 注意：要使用该插件必须得让 ctags 软件达到 5.8.1 版本，ctags 官网上并无该版本，可在 <http://dfrank.ru/ctags581/en.html> 下载，安装后可用 `ctags --version` 确认下版本是否正确。

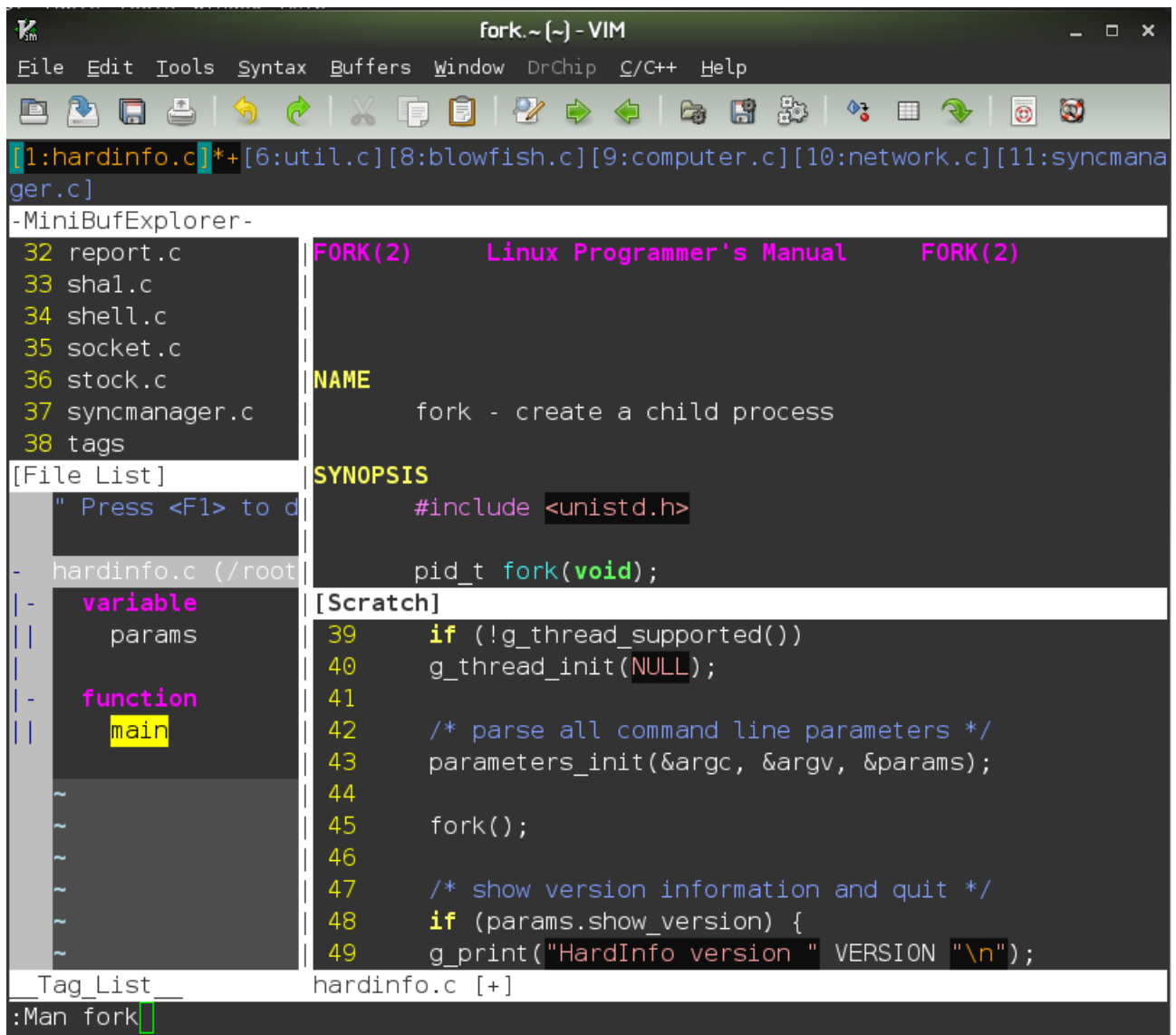
[- 系统函数调用参考 -]

有过 Win32 SDK 开发经验的朋友对 MSDN 或多或少有些迷恋吧，有些 API 多达 7、8 个参数，有套函数功能描述、参数讲解、返回值说明的文档那是很有必要滴。别急，vim 也能做到。

- 插件名：man.vim (内置)
- 操作：先在 vim 中启动该插件:`source $VIMRUNTIME/ftplugin/man.vim` (可以加入.vimrc 中自动启动该插件)，需要查看系统函数参考时输入:`Man sys_api`即可在新建分割子窗口中查看到 `sys_api()`函数参考信息
- 注意：要使用该功能，系统中必须先安装 man-pages 和 man-pages-posix 两套帮助手册。

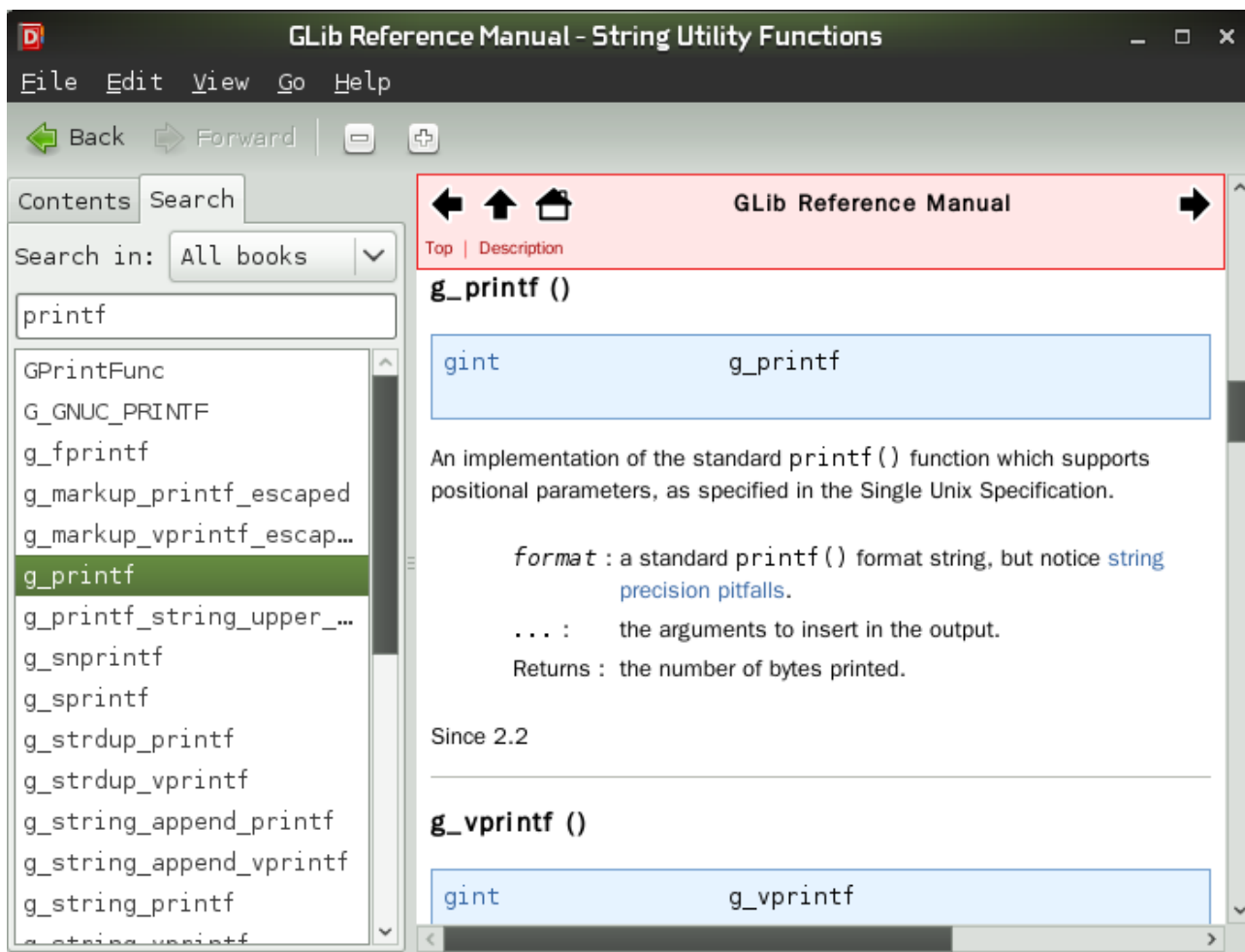


(图 6 : 库函数调用参考)



(图 7：系统函数调用参考)

此外，如果你从事 gnome 开发，还可以安装独立软件 devhelp，这是 GTK 版的 MSDN。如下图所示：



(图 8 : devhelp)

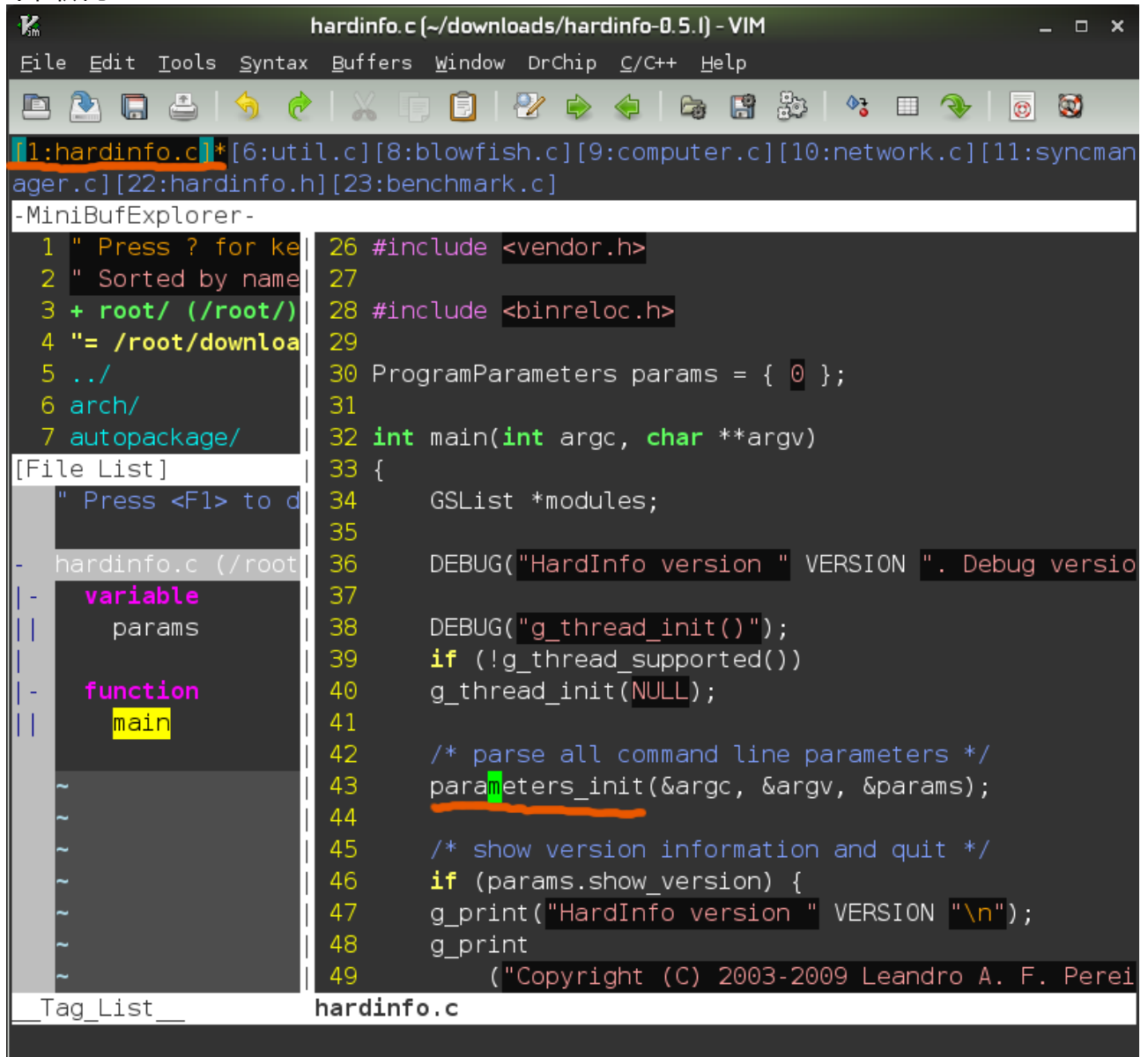
[- 快速查看定义 -]

上面说了如何查看系统函数调用，自己写的函数又如何快速查看其函数定义了？单代码文件的项目倒是不麻烦，无非上下拖动下滚动条或者多按几次j、k键而已，对于动辄十多个代码文件的一般项目来说，准确、快速查找到函数定义对于提高开发效率非常有帮助。下面介绍如何快速跳转到函数定义、变量定义、结构定义、成员定义处。

快速跳转到函数定义处，也是通过标签来实现的，所以必须得有tags文件支持，如果你还没实现前面“智能补全”部分的功能，那请倒回去看看，成功后再继续这部分内容，如果已经实现，你可以随便找个自定义函数（注，一定要是自定义函数，因为ctags未对系统函数生成标签，对于系统函数，我一般只关注其功能、参数、返回值等信息，不会关注起实现或称为定义。前者通过“系统函数调用参考”部分已经实现，后者如果你的确需要，可将“ctags -R.”替换成“ctags -R --fields=+IS /usr/include”，“/usr/include”为系统函数头文件和实现文件所在目录），把光标移到上面，输入CTRL-]，呵呵，是不是奇迹发生了呀。

比如，在hardinfo.c文件中调用函数parameters_init()，将光标移到该函数上，如

下图所示：

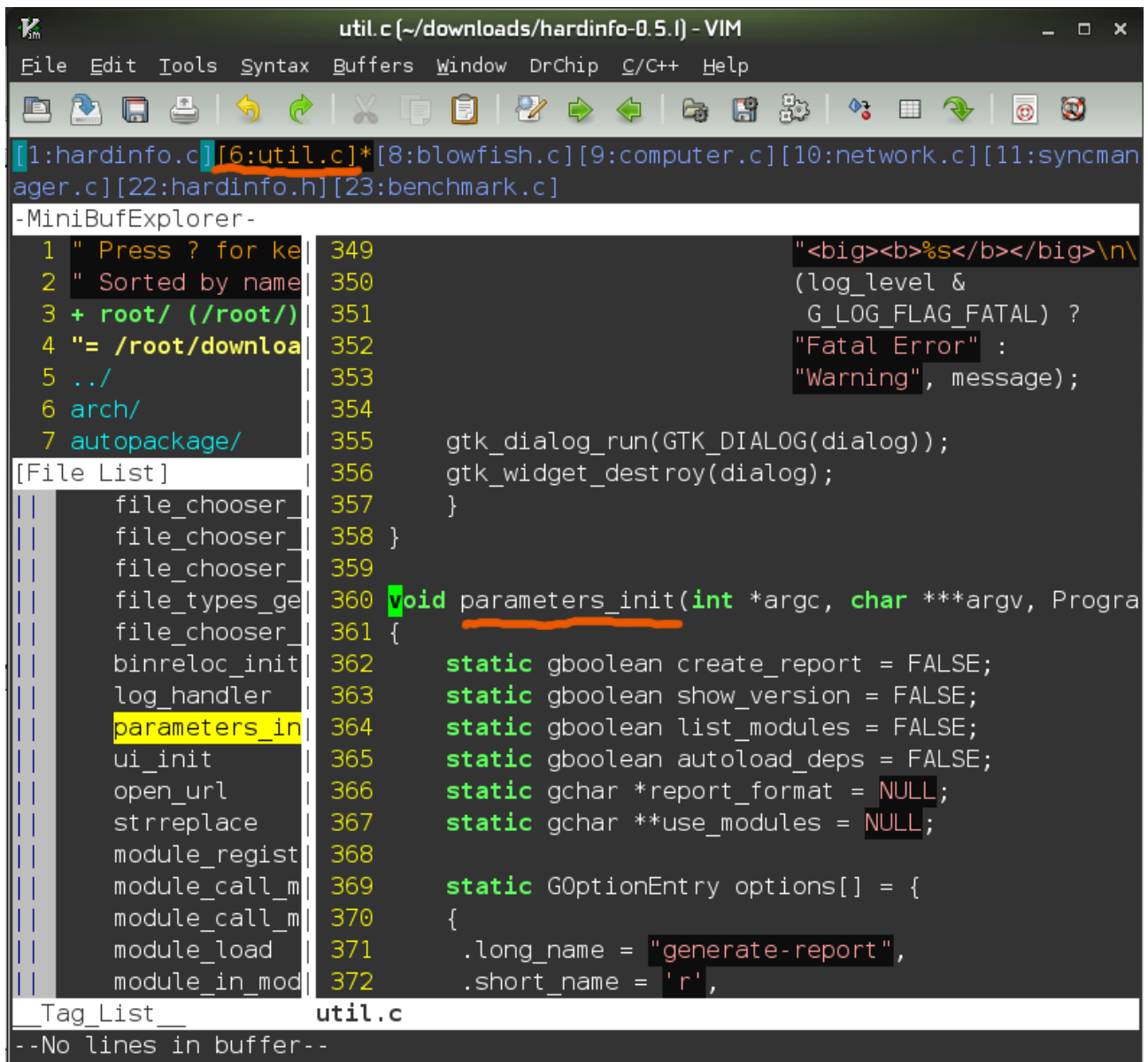


```
hardinfo.c (~/.downloads/hardinfo-0.5.1) - VIM
File Edit Tools Syntax Buffers Window DrChip C/C++ Help

[1:hardinfo.c]*[6:util.c][8:blowfish.c][9:computer.c][10:network.c][11:syncman
ager.c][22:hardinfo.h][23:benchmark.c]
-MiniBufExplorer-
1 " Press ? for ke
2 " Sorted by name
3 + root/ (/root/)
4 "= /root/downloa
5 ../
6 arch/
7 autopackage/
[File List]
" Press <F1> to d
- hardinfo.c (/root
|- variable
||   params
|- function
||   main
~
~
~
~
~
~
~
~
~
~
Tag_List__
hardinfo.c
26 #include <vendor.h>
27
28 #include <binreloc.h>
29
30 ProgramParameters params = { 0 };
31
32 int main(int argc, char **argv)
33 {
34     GSList *modules;
35
36     DEBUG("HardInfo version " VERSION ". Debug versio
37
38     DEBUG("g_thread_init()");
39     if (!g_thread_supported())
40         g_thread_init(NULL);
41
42     /* parse all command line parameters */
43     parameters_init(&argc, &argv, &params);
44
45     /* show version information and quit */
46     if (params.show_version) {
47         g_print("HardInfo version " VERSION "\n");
48         g_print
49             ("Copyright (C) 2003-2009 Leandro A. F. Perei
```

(图 9：准备跳转到函数定义处)

输入 CTRL-J 后，通过 tags 文件，vim 找到 parameters_init() 定义在 util.c 文件的 360 行，并自动定位到该文件的正确行数上，如下图所示：



```
util.c [~/downloads/hardinfo-0.5.1] - VIM
File Edit Tools Syntax Buffers Window DrChip C/C++ Help

[1:hardinfo.c][6:util.c]*[8:blowfish.c][9:computer.c][10:network.c][11:syncman
ager.c][22:hardinfo.h][23:benchmark.c]

-MiniBufExplorer-
1 " Press ? for ke 349 "<big><b>%s</b></big>\n\
2 " Sorted by name 350 (log_level &
3 + root/ (/root/) 351 G_LOG_FLAG_FATAL) ?
4 "= /root/downloa 352 "Fatal Error" :
5 ../ 353 "Warning", message);
6 arch/ 354
7 autopackage/ 355 gtk_dialog_run(GTK_DIALOG(dialog));
[File List] 356 gtk_widget_destroy(dialog);
|| file_chooser_ 357 }
|| file_chooser_ 358 }
|| file_chooser_ 359
|| file_types_ge 360 void parameters_init(int *argc, char ***argv, Progra
|| file_chooser_ 361 {
|| binreloc_init 362 static gboolean create_report = FALSE;
|| log_handler 363 static gboolean show_version = FALSE;
|| parameters_in 364 static gboolean list_modules = FALSE;
|| ui_init 365 static gboolean autoload_deps = FALSE;
|| open_url 366 static gchar *report_format = NULL;
|| strreplace 367 static gchar **use_modules = NULL;
|| module_regist 368
|| module_call_m 369 static GOptionEntry options[] = {
|| module_call_m 370 {
|| module_load 371 .long_name = "generate-report",
|| module_in_mod 372 .short_name = 'r',
__Tag_List__ util.c
--No lines in buffer--
```

(图 10 : 正确跳转到函数定义处)

OK , 上面演示了如何快速跳转到函数定义处 , 变量、结构、成员也是类似的。另外 , CTRL -] 跳转到定义处后。如果此时你还想再跳回先前位置 , 按 CTRL-O , 前进按 CTRL-I。

此外 , 有时我们在阅读代码时 , 希望有个窗口能将当前代码文件中所有函数名、变量名、结构名等以列表形式罗列出来 , 以便有针对性的分析代码。呵呵 , 可以滴 ~。

- 插件名 : taglist 插件
- 操作 : 输入:Tlist 即可调出名字列表子窗口 , 切换到不同代码文件时 , 列表会自动更新
- 注意 : 请在.vimrc 中增加如下配置信息 :

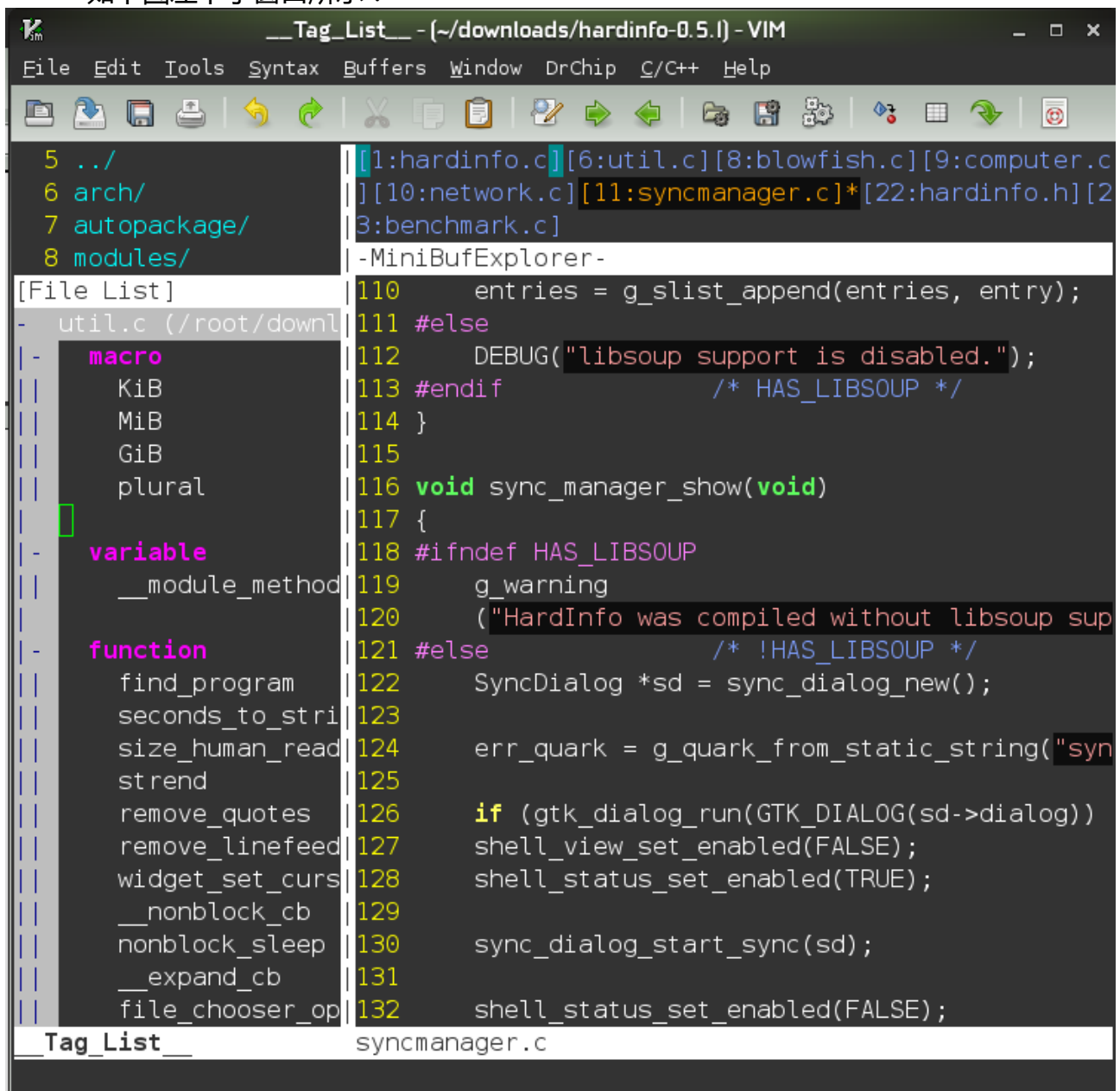
```
"定义快捷键的前缀 , 即<Leader>
let mapleader=";"
"设置 tablist 插件只显示当前编辑文件的 tag 内容 , 而非当前所有打开文件的 tag 内容
```

```

let Tlist_Show_One_File=1
"设置显示标签列表子窗口的快捷键。速记：tag list
nnoremap <Leader>tl :TlistToggle<CR>
"设置标签子窗口的宽度
let Tlist_WinWidth=20
"标签列表窗口显示或隐藏不影响整个 gvim 窗口大小
let Tlist_Inc_Winwidth=0

```

如下图左下子窗口所示：



(图 11：标签列表)

[- 语法高亮 -]

现在已是千禧年后的十年了，早已告别上世纪六、七十年代黑底白字的时代，如果编码时没有语法高亮，肯定会让你的代码失去活力，即使在字符模式下编程（感谢伟大的 fbterm），我也会开启语法高亮功能。

vim 自身就支持语法高亮，只须打开相关设置即可。请将如下配置信息添加到.vimrc：

```
"打开语法高亮
set syntax enable
"允许按指定主题进行语法高亮，而非默认高亮主题
set syntax on
"指定配色方案
colorscheme evening
```

我选用内置的 evening 配色方案，灰底、白字、黄色关键字、蓝色注释.....，效果还不错，比较适合我的审美观。如果不满意，vim 还提供了 10 多种配色方案供你选择，GUI 下，可以通过菜单（Edit -> Color Scheme）试用不同方案，字符模式下，需要你手工调整配置信息，再重启 vim 查看效果（推荐 csExplorer.vim 插件，可在字符模式下不用重启即可查看效果）。

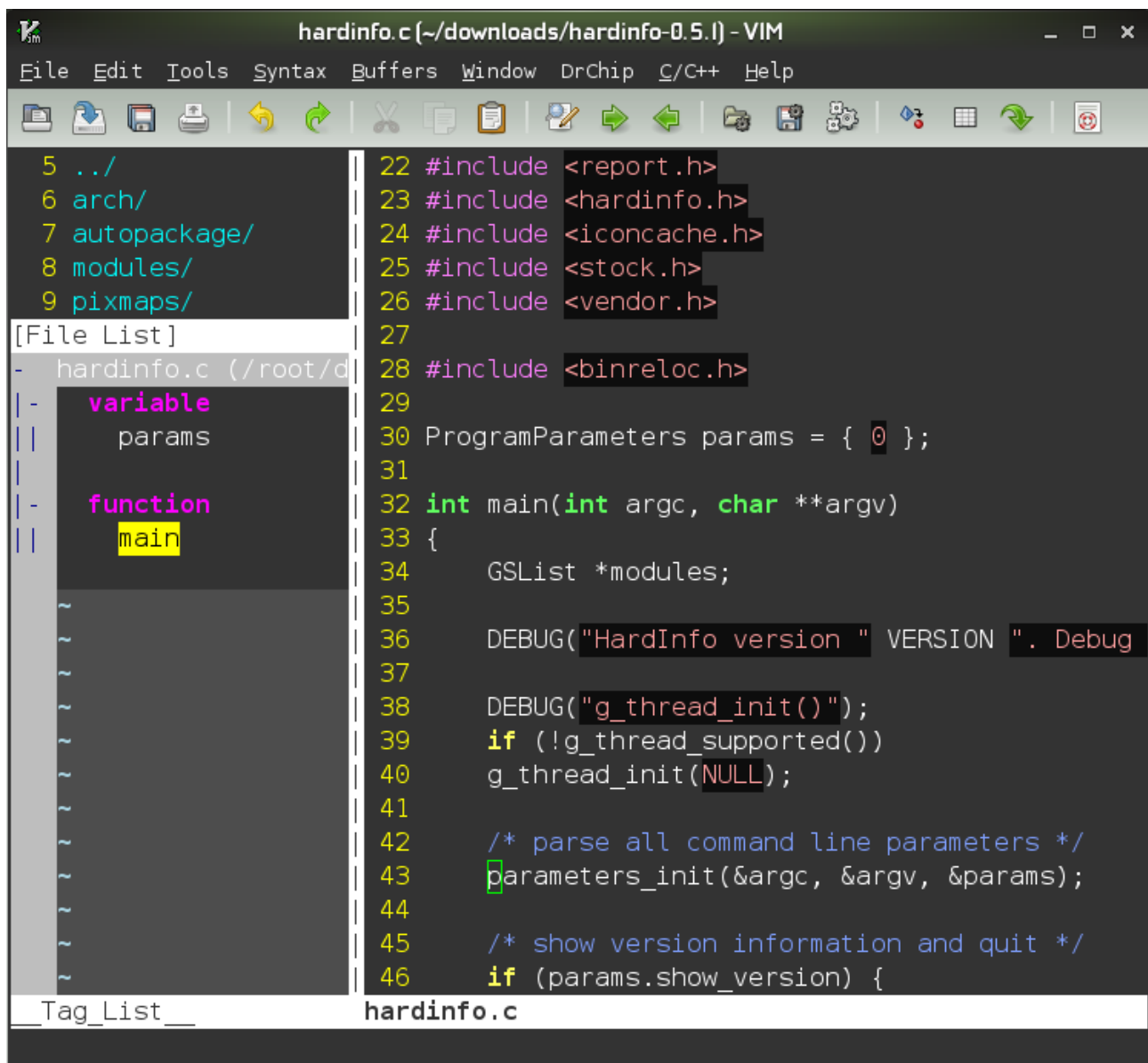
【 - 多文档编辑 - 】

一个 EXCEL 文档可以有多个 SHEET，你可以在不同 SHEET 间来回切换，同样，编程时也需要类似功能，即，同时打开多个文件，可以自由自在地在不同代码文件间游历。这种需求，vim 是通过 buffer 来实现的。每打开一个文件 vim 就对应创建一个 buffer，多个文件就有多个 buffer。

- 插件名：MiniBufExplorer
- 操作：打开一个以上文档时，vim 在窗口顶部自动创建 buffer 列表窗口。光标在任何位置时，CTRL-TAB 正向遍历 buffer，CTRL-SHIFT-TAB 逆向遍历；光标在 MiniBufExplorer 窗口内，输入 d 删除光标所在的 buffer
- 注意：将如下信息加入.vimrc 中：

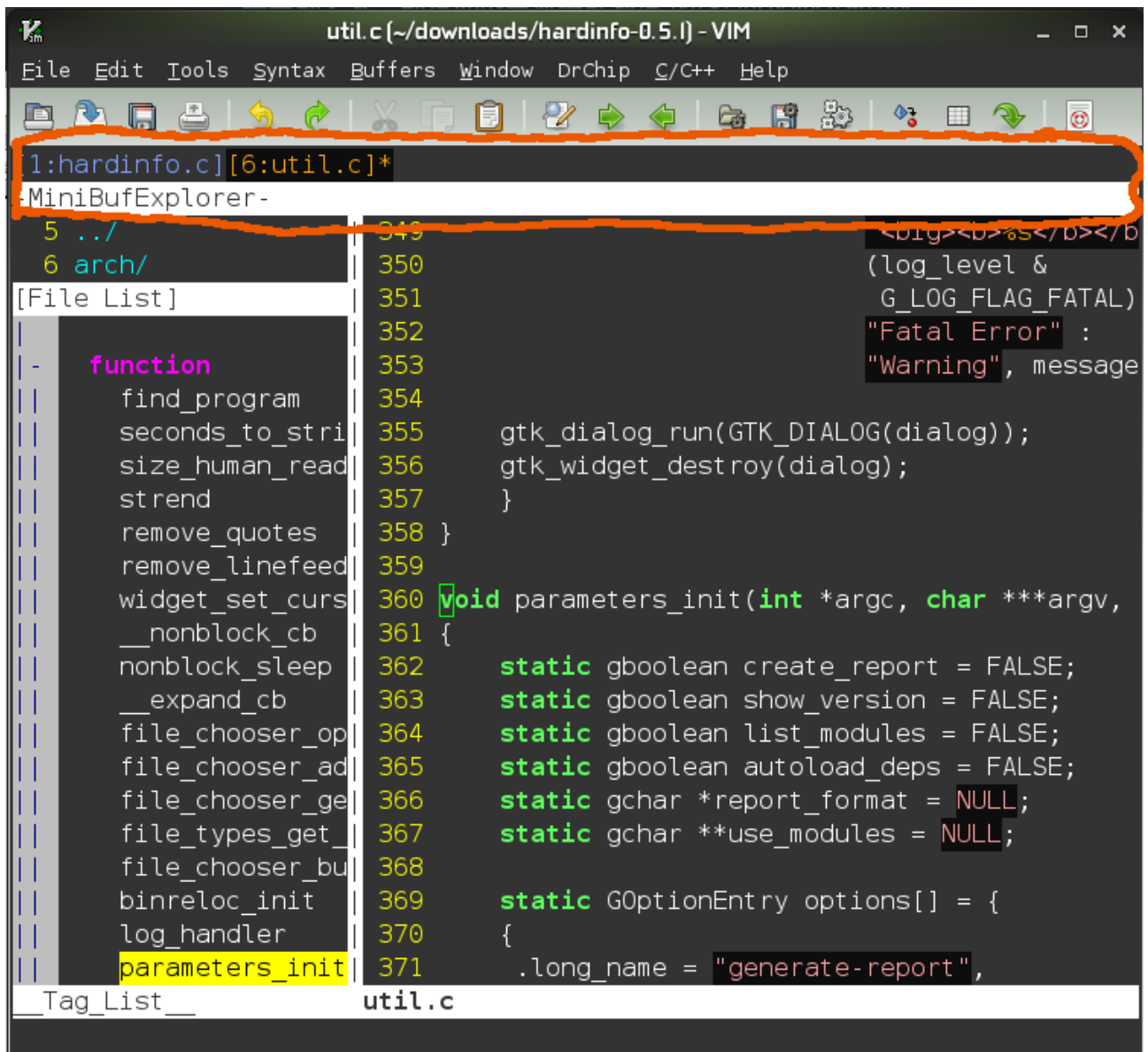
```
"允许光标在任何位置时用 CTRL-TAB 遍历 buffer
let g:miniBufExplMapCTabSwitchBufs=1
```

编辑单个文档时，不会出现 buffer 列表，如下图所示：



(图 12：编辑单个文档)

当前编辑的是 hardinfo.c 文件，该文件中有调用 parameters_init() 函数，但该函数定义在 util.c 文件中，当我在 parameters_init() 上输入 CTRL-] 后，vim 新建 util.c 文件的 buffer 并自动定位到 parameters_init()，这时，vim 同时在编辑 hardinfo.c 和 util.c 两个文档，可从顶部的 buffer 列表中查看到。如下图所示：



(图 13：同时编辑多个文档)

[- 代码折叠 -]

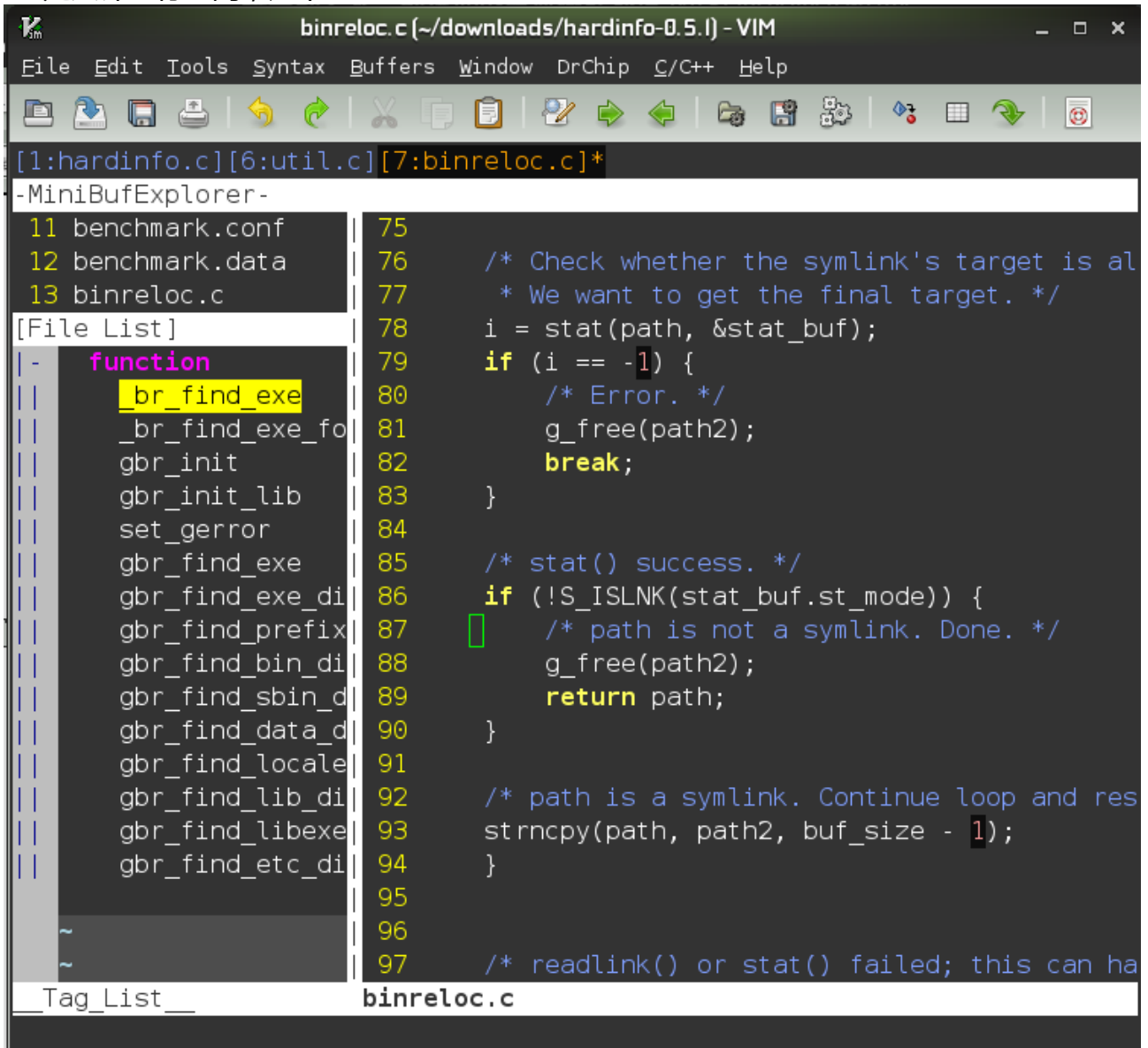
有时为了去除干扰，集中精力在某部分代码段上，我会把不关注部分代码折叠起来。vim 自身支持多种折叠，包括：手动建立折叠（manual）、相同缩进距离的行构成折叠（indent）、'foldexpr' 给出每行的折叠（expr）、标志用于指定折叠（marker）、语法高亮项目指定折叠（syntax）、没有改变的文本构成折叠（diff）。用于编程时的折叠当然就选“语法高亮项目指定折叠（syntax）”啦。

- 操作：za，打开或关闭当前折叠；zM，关闭所有折叠；zR，打开所有折叠
- 注意：在.vimrc 中增加如下信息即可实现代码折叠：

```
"选择代码折叠类型
set foldmethod=syntax
"启动 vim 时不要自动折叠代码
```

```
set foldlevel=100
```

如，当前光标位于 87 行的 if 语句块内，该语句块处于正常展开的状态，占据从 86 到 90 处共计 5 行空间，如下：



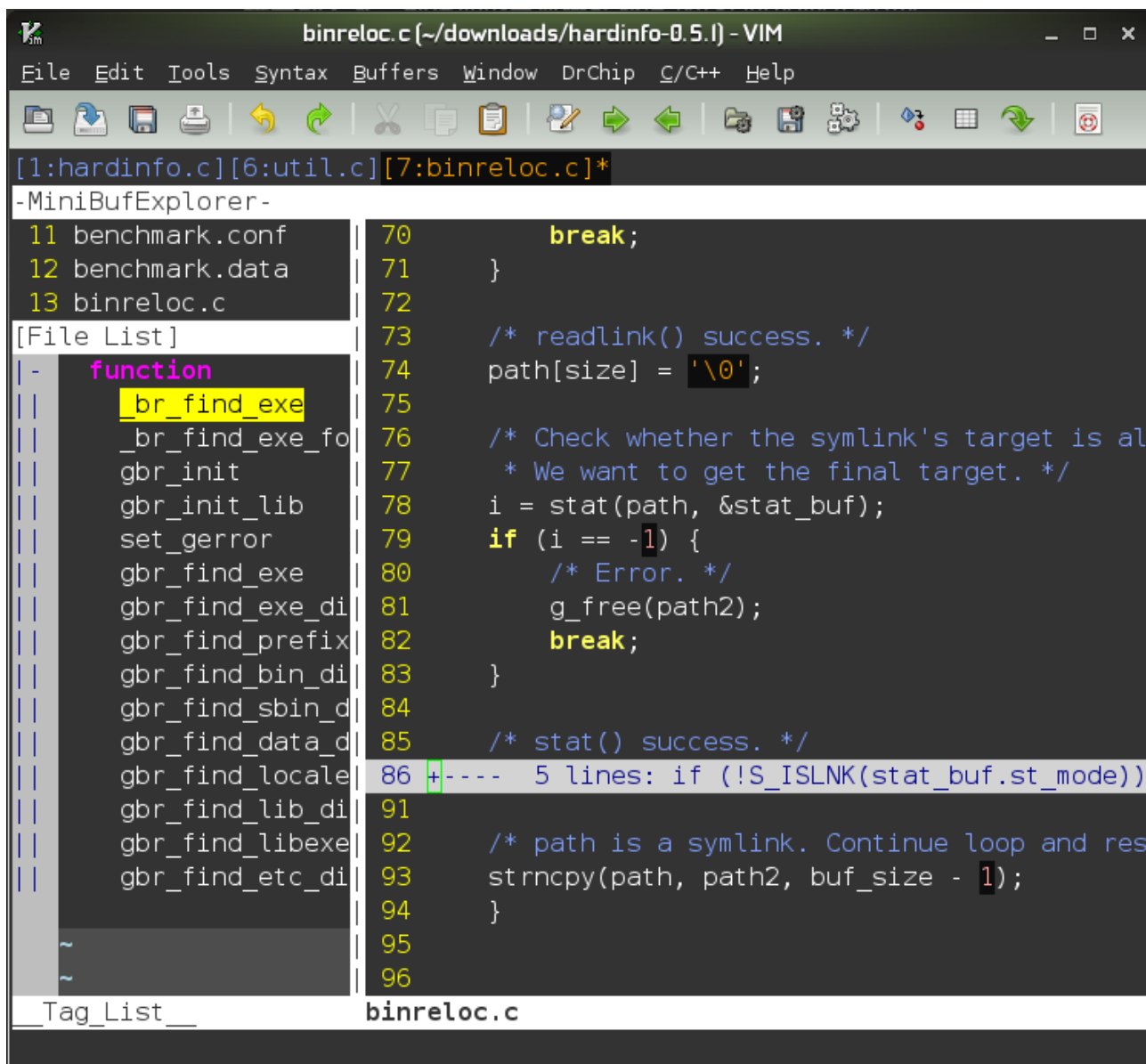
```
binreloc.c (~/downloads/hardinfo-0.5.1) - VIM
File Edit Tools Syntax Buffers Window DrChip C/C++ Help

[1:hardinfo.c][6:util.c][7:binreloc.c]*
-MiniBufExplorer-
11 benchmark.conf
12 benchmark.data
13 binreloc.c
[File List]
- function
| _br_find_exe
| _br_find_exe_fo
| gbr_init
| gbr_init_lib
| set_gerror
| gbr_find_exe
| gbr_find_exe_di
| gbr_find_prefix
| gbr_find_bin_di
| gbr_find_sbin_d
| gbr_find_data_d
| gbr_find_locale
| gbr_find_lib_di
| gbr_find_libexe
| gbr_find_etc_di
| ~
| ~
Tag_List binreloc.c

75
76 /* Check whether the symlink's target is al
77 * We want to get the final target. */
78 i = stat(path, &stat_buf);
79 if (i == -1) {
80 /* Error. */
81 g_free(path2);
82 break;
83 }
84
85 /* stat() success. */
86 if (!S_ISLNK(stat_buf.st_mode)) {
87 /* path is not a symlink. Done. */
88 g_free(path2);
89 return path;
90 }
91
92 /* path is a symlink. Continue loop and res
93 strncpy(path, path2, buf_size - 1);
94 }
95
96
97 /* readlink() or stat() failed; this can ha
```

(图 14：正常展开的代码)

输入 za 后，这 5 行合并为一行，显示第一行的内容，如下所示：



(图 15：折叠后的代码)

[- 工程内查找与替换 -]

有个名为 iFoo 的全局变量，被工程中 10 个文件引用过，由于你岳母觉得匈牙利命名法严重、异常、绝对以及十分万恶，为讨岳母欢心，不得不将该变量更名为 foo，怎么办？依次打开每个文件，逐一查找后替换？

vim 既然被称为“编辑器之神”，这点请求还是可以满足嘛^_^。Vim 自身支持全局替换：先选择要替换的文档:args 1.c 2.c ... 10.c，执行全局替换命令:argdo %s/\<iFoo\>/foo/ge | update，其中，iFoo 将被 foo 替代，e 表示忽略错误，update 用于保存那些有被执行替换操作的文档。

此外，要进行工程内全局查找，可以借助插件实现。

- 插件名：grep 插件
- 操作：在空白处输入“;sp”，grep 插件提示输入待查找关键字后回车即可执行搜索；

如果在非空白处输入 “;sp” ，grep 插件自动将当前光标所在单词作为关键字进行搜索。搜索结果将罗列在 quickfix 中（注，vim 与很多外部命令、插件的交互信息都将在 quickfix 中呈现，这里说到的搜索结果是一个例子，另外一个著名例子为 gcc 的输出信息。可用:cw 命令打开或关闭 quickfix 窗口）

- 注意：在.vimrc 中，增加如下配置信息：

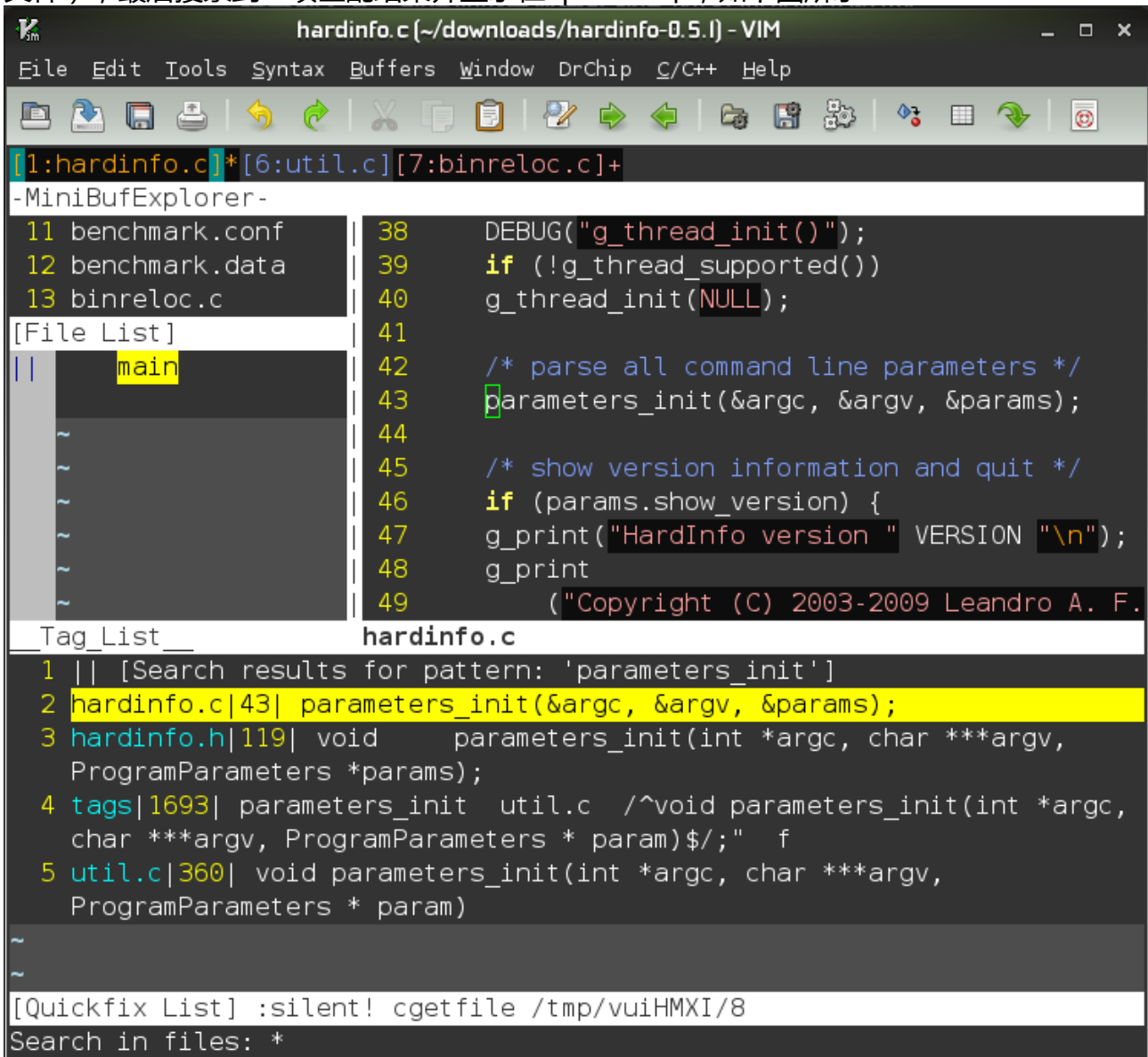
"定义快捷键关闭当前分割窗口

nmap <Leader>q :q<CR>

"使用 Grep.vim 插件在工程内全局查找，设置快捷键。快捷键速记法：search in project

nnoremap <Leader>sp :Grep<CR>

比如，光标移到 parameters_init 下，输入;sp 后，grep 插件自动提取 parameters_init 为搜索关键字，并给出在哪些类型的文件中内搜索（默认为该目录下所有文件），最后搜索到 4 项匹配结果并显示在 quickfix 中，如下图所示：



```
hardinfo.c[~/downloads/hardinfo-0.5.1] - VIM
File Edit Tools Syntax Buffers Window DrChip C/C++ Help

[1:hardinfo.c]*[6:util.c][7:binreloc.c]+
-MiniBufExplorer-
11 benchmark.conf
12 benchmark.data
13 binreloc.c
[File List]
|| main
~
~
~
~
~
~
Tag_List__ hardinfo.c
1 || [Search results for pattern: 'parameters_init']
2 hardinfo.c|43| parameters_init(&argc, &argv, &params);
3 hardinfo.h|119| void parameters_init(int *argc, char ***argv,
ProgramParameters *params);
4 tags|1693| parameters_init util.c /^void parameters_init(int *argc,
char ***argv, ProgramParameters * param)$/;" f
5 util.c|360| void parameters_init(int *argc, char ***argv,
ProgramParameters * param)
~
~
[Quickfix List] :silent! cgetfile /tmp/vuiHMXI/8
Search in files: *
```

(图 16 : 工程内查找)

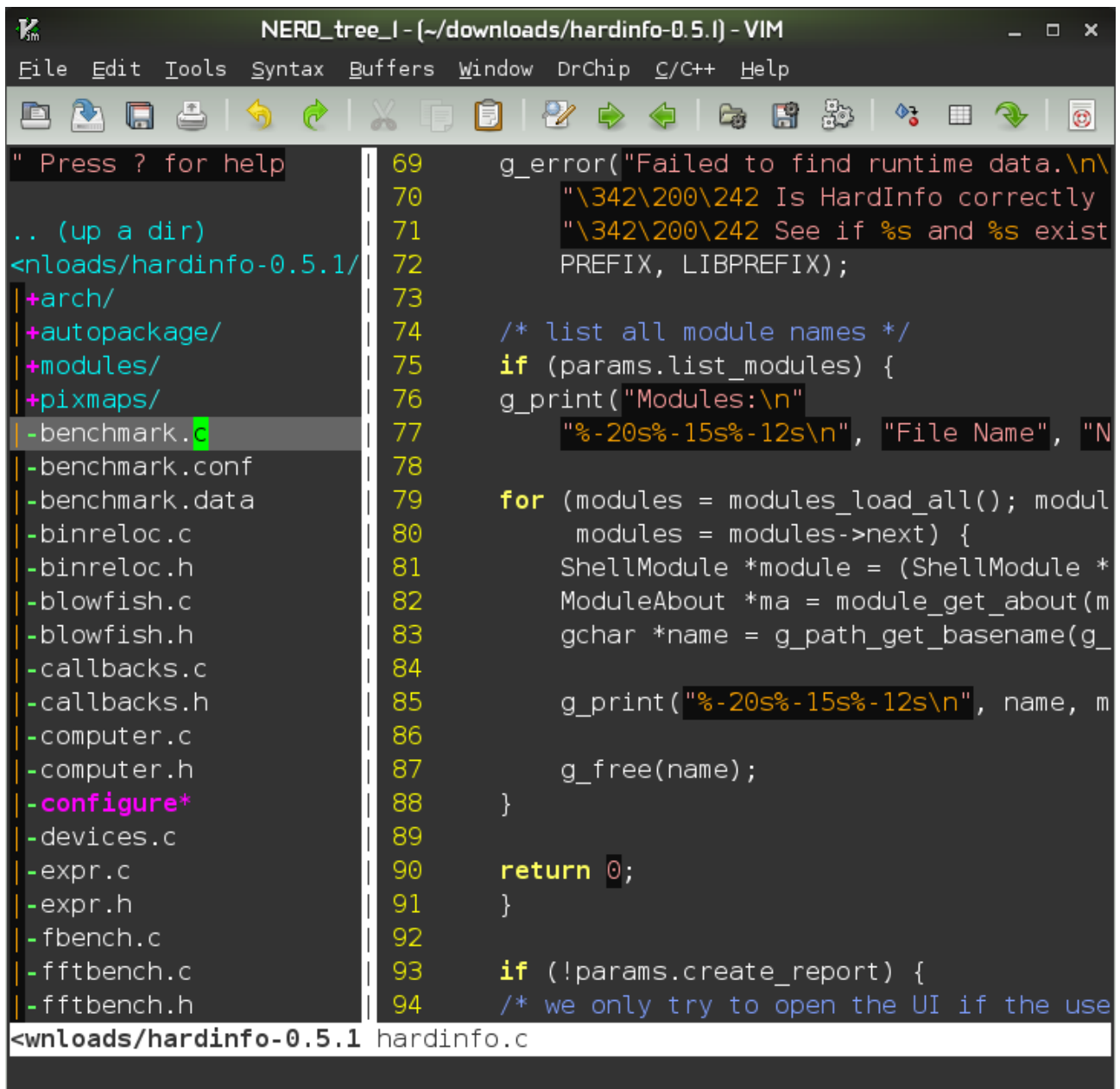
[- 工程文件浏览 -]

我通常将工程相关的文档放在同个目录下，通过 NERDtree 插件查看文件列表，要打开哪个文件，光标选中后回车即可在新 buffer 中打开。

- 插件：NERDtree.vim
- 操作：回车，打开文档；r，刷新工程目录文件列表；I，显示或隐藏隐藏文件
- 注意：请将如下信息加入.vimrc 中：

```
"使用 NERDTree 插件查看工程文件。设置快捷键，速记法：file list  
nmap <Leader>fl :NERDTreeToggle<CR>  
"设置 NERDTree 子窗口宽度  
let NERDTreeWinSize=23
```

键入“ ;fl” 后，左边子窗口为工程项目文件列表，如下图所示：



The screenshot shows a Vim editor window titled "NERD_tree_1 - (~/.downloads/hardinfo-0.5.1) - VIM". The menu bar includes File, Edit, Tools, Syntax, Buffers, Window, DrChip, C/C++, and Help. The toolbar contains various icons for file operations and editing. The left pane displays a file tree for the directory "~/.downloads/hardinfo-0.5.1/". The tree includes files like benchmark.c, benchmark.conf, benchmark.data, binreloc.c, binreloc.h, blowfish.c, blowfish.h, callbacks.c, callbacks.h, computer.c, computer.h, configure*, devices.c, expr.c, expr.h, fbench.c, fftbench.c, and fftbench.h. The right pane shows C code from the file hardinfo.c, with line numbers 69 through 94. The code includes error handling, module listing, and a loop for loading modules. The status bar at the bottom shows "<wnloads/hardinfo-0.5.1 hardinfo.c".

```
" Press ? for help
.. (up a dir)
<nloads/hardinfo-0.5.1/
|+arch/
|+autopackage/
|+modules/
|+pixmap/
|-benchmark.c
|-benchmark.conf
|-benchmark.data
|-binreloc.c
|-binreloc.h
|-blowfish.c
|-blowfish.h
|-callbacks.c
|-callbacks.h
|-computer.c
|-computer.h
|-configure*
|-devices.c
|-expr.c
|-expr.h
|-fbench.c
|-fftbench.c
|-fftbench.h
<wnloads/hardinfo-0.5.1 hardinfo.c

69     g_error("Failed to find runtime data.\n\
70         "\342\200\242 Is HardInfo correctly
71         "\342\200\242 See if %s and %s exist
72         PREFIX, LIBPREFIX);
73
74     /* list all module names */
75     if (params.list_modules) {
76         g_print("Modules:\n"
77             "%-20s%-15s%-12s\n", "File Name", "N
78
79     for (modules = modules_load_all(); modul
80         modules = modules->next) {
81         ShellModule *module = (ShellModule *
82         ModuleAbout *ma = module_get_about(m
83         gchar *name = g_path_get_basename(g_
84
85         g_print("%-20s%-15s%-12s\n", name, m
86
87         g_free(name);
88     }
89
90     return 0;
91 }
92
93 if (!params.create_report) {
94     /* we only try to open the UI if the use
```

(图 17：项目文件列表)

[- 暂未实现功能 -]

好了，以上功能基本达到我对 IDE 的期望值了，满足了？没有，还有几个问题现在还没很好地解决，或者是有思路了，还没来得及实践，记录下来，空了继续探索。如果你清楚，麻烦写信告诉我（yangyang.gnu@gmail.com），多谢多谢 @_@

- 1、一键编译、链接、运行程序。
- 2、集成调试器，实现源码级调试。思路：Pyclewn 插件
- 3、重启 vim 时恢复到上次编辑环境。包括打开的文件、BUFFER、光标位置等。思路：通过 vim 的会话文件实现。

[- 附一：.vimrc 信息汇总 -]

以下信息源自我的 ~/.vimrc 文件，主要对 vim 自身进行个性化配置、对各插件进行配置以及快捷键设置，每个配置项都有对应注释，可根据你自己情况按需择取。

[illegible]

"搜索时大小写不敏感

set ignorecase

"在命令行显示当前输入的命令

set showcmd

"禁止折行

set nowrap

"关闭兼容模式

set nocompatible

"禁止显示滚动条

set guioptions-=l

set guioptions-=L

set guioptions-=r

set guioptions-=R

"开启文件类型侦测

filetype on

"根据侦测到的不同类型加载对应的插件

filetype plugin on

"根据侦测到的不同类型采用不同的缩进格式

filetype indent on

"定义快捷键的前缀，即<Leader>

let mapleader=";"

"定义快捷键到行首和行尾

nmap lh 0

nmap le \$

"定义快捷键关闭当前分割窗口

nmap <Leader>q :q<CR>

"定义快捷键保持当前窗口内容

nmap <Leader>w :w<CR>

"设置快捷键将选中文本块复制至系统剪贴板


```
"noremap <Leader>y "+y  
"设置快捷键将系统剪贴板内容粘贴至 vim  
nmap <Leader>p "+p  
  
"<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<  
"  
">>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>  
">>>插件相关配置  
  
"使用 NERDTree 插件查看工程文件。设置快捷键，速记：file list  
nmap <Leader>fl :NERDTreeToggle<CR>  
"设置 NERDTree 子窗口宽度  
let NERDTreeWinSize=20  
"设置 NERDTree 子窗口位置  
let NERDTreeWinPos="right"  
  
"设置 tablist 插件只显示当前编辑文件的 tag 内容，而非当前所有打开文件的 tag 内容  
let Tlist_Show_One_File=1  
"设置显示标签列表子窗口的快捷键。速记：tag list  
nnoremap <Leader>tl :TlistToggle<CR>  
"设置标签子窗口的宽度  
let Tlist_WinWidth=20  
"标签列表窗口显示或隐藏不影响整个 gvim 窗口大小  
let Tlist_Inc_Winwidth=0  
  
"使用 minibufexpl.vim 插件管理 buffer，设置允许光标在任意位置时，通过 CTRL-TAB 遍历 buffer  
let g:minibufExplMapCTabSwitchBufs=1  
  
"支持用\K 查看 SHELL 命令和 C 库函数的 man 信息  
source $VIMRUNTIME/ftplugin/man.vim  
  
"取消补全内容以分割子窗口形式出现，只显示补全列表  
set completeopt=longest,menu  
  
"使用 Grep.vim 插件在工程内全局查找，设置快捷键。快捷键速记法：search in project  
nnoremap <Leader>sp :Grep<CR>  
  
"设置快捷键 qs 遍历各分割窗口。快捷键速记法：goto the next spilt window
```

```
nnoremap <Leader>gs <C-W> <C-W>
```

使用 new-omni-completion 插件智能补全代码。该插件默认使用 CTRL-X CTRL-O 补全函数名或变量名，自定义快捷键为 TAB

```
imap <Leader> <TAB> <C-X> <C-O>
```

"VIM 支持多种文本折叠方式，我 VIM 多用于编码，所以选择符合编程语言语法的代码折叠方式。

```
set foldmethod=syntax
```

"启动 vim 时打开所有折叠代码。foldlevel 用于设置闭合折叠代码的层级

```
set foldlevel=100
```

[illegible]

[- 附二： vim 常用操作汇总 -]

我常用的 vim 操作记录如下，备忘~_~

//最后更新时间：2011年7月24日12点41分

[- 移动 -]

- 跳转到光标先前位置、下个位置：CTRL - O、CTRL - I
- 以单词为单位移动光标：w、b、W、B
- 翻页：CTRL - F、CTRL - B
- 整个文本中移动光标：gg、G、数字G、数字%
- 当前页中移动光标：H、M、L
- 移动光标所在行的位置：zz、zt、zb

[- 搜索 -]

- 大小写不敏感: `:set ignorecase`, 大小写敏感: `:set noignorecase`
- 行内搜索: `fx`。X 代表要搜索的单个文字 (也可以是汉字)。FX 为方向搜索。分号重复, 逗号反方向重复
- 整词搜索: `\<word\>`。整词首尾可拆分搜索
- 行首、行尾搜索: `/^word`、`/word$`
- 搜索替代字符: `/ab.de`。 “.” 代表任意一个字符
- 替换: `:1,$s/a/b/g`

[- 自动化命令 -]

- ◆ 重复上次命令：..

- ◆ 撤销上步操作：`u`
- ◆ 重复上步操作：`CTRL - R`。于不同，`CTRL - R` 对命令历史记录进行进栈 / 出栈操作

[- 分割窗口 -]

- ◆ 新建空白分割窗口：`:new`
- ◆ 在新建空白分割窗口中打开指定文件：`:split filename`
- ◆ 在新建空白分割窗口中显示当前分割窗口内容：`:split`
- ◆ 分割窗口高度调整。增加一行：`CTRL-W, SHIFT-+`；减少一行：`CTRL-W, -`；增加到最大高度：`CTRL-W, SHIFT--`；调整到指定高度：`heightCTRL-W, SHIFT--`

[- 其他 -]

- ◆ 取消上次搜索结果高亮显示：`:nohlsearch`
- ◆ 文本另存为：`:saveas file.txt`
- ◆ 多段文本复制：使用寄存器，`"ay2j`，`"ap`，其中，双引号为寄存器引用前缀，`a` 为自定义寄存器名（只能为一个字母或数字，或代表系统剪贴板的 `+`）
- ◆ 宏记录：使用寄存器，`qb -> 操作 -> q`，其中，`q` 为宏记录开始与结束命令，`b` 为寄存器，宏回放使用 `@b`。宏回放可加计数器前缀。可 `"bp` 打印宏内容，编辑后再 `"bY`。注：复制粘贴和宏记录使用同一套寄存器，所以，同个寄存器的内容即可用于粘贴，也可视为宏记录
- ◆ 选择文本块：`v`、`V`、`CTRL - V`。`o`、`O` 移动光标在文本块四个角的位置。用 `I` 或 `A` 命令编辑第一行，再恢复到普通模式下时，被选择块每行首或尾都会有相同新增内容；`r` 命令单个字符替换文本块
- ◆ 操作计数器：数字 - 操作
- ◆ 在线帮助：`help keywords`
- ◆ 匹配括号：`%`
- ◆ 恢复选项的默认值：`set option&`
- ◆ 字母大小写转换：`~`
- ◆ 转换为 html 文件：`:source $VIMRUNTIME/syntax/2html.vim , :write main.c.html`
- ◆ 在线加载配置文件或插件：`:source filepath`。如，重新加载配置文件：`:source ~/.vimrc`
- ◆ 删除光标所在字符到行尾的内容：`D`
- ◆ 快速向下查找光标所在字符串：`*`；向上：`#`
- ◆ 格式化代码：`=`、`>>`、`<<`
- ◆ Vim 会在你连续 4 秒不键入内容时跟磁盘同步一次（内容写入 vim 临时文件中），或者是连续键入了 200 个字符之后。这可以通过 `'updatetime'` 和 `'updatecount'` 两个选项来控制。
- ◆ VIM 提供两种方式执行外部命令，一种是 `!:cmd`，一种是 `!cmd`，前者完全等同于在 shell 中执行命令，后者相当于同时对命令输入输出重定向，即，将选中的文本块内容作为输入传递给外部命令并用执行结果替换选中文本块。后者用途较为广泛，如，对文本内容排序，可先选中待排序文本块，再键入 `!sort` 即可，注意，不要键入冒号。也可以仅重定向外部命令输出，即，读取外部命令执行结果：`:read !ls`，将 `ls` 命令执行结果插入当前行。也可以仅重定向外部命令输入，即，将选中文本输入给外部命令执行：`:write !wc`，将对选中文本块进行计数操

作。

- 查看 man 信息：先执行:source \$VIMRUNTIME/ftplugin/man.vim。光标移到待查看命令下后键入“\K”后即可在新子窗口中看到 man 内容，或者” :Man cmd”
- VIM 支持命令行补全，查看全部可键入 CTRL-D。如，键入:set i后键入 CTRL-D 则显示 set 命令支持的所有以 i 开头的选项
- 命令历史窗口：q:，移动光标到指定行回车即可执行该行命令
- 直接打开文件：键入 gf，VIM 将当前光标所在字符串视为文件路径并尝试打开编辑该文件。若是绝对路径，则直接打开，若是相对路径，VIM 在 path 选项指定的路径范围内进行查找，该 path 为 VIM 的一个选项而非 SHELL 的环境变量，默认为和/usr/include，可通过:set path+=addpath或:set path-=removepath来增删路径。注：若要在分割子窗口中打开可以 CTRL-W f
- 重新选中上次选择的文本块：gv
- 选中结对符内的字符串：如，va{。{只是一种结对符，可自行替换为其他结对符
- 清空结对符内的字符串：如，di”。”只是一种结对符，可自行替换为其他结对符
- 安装 vim 中文帮助 <http://vimcdoc.sourceforge.net/>
- 显示当前光标在文档中的位置信息：CTRL-G