

(vim) 启动选项

手稿

封面

当使用 vim 命令启动编辑会话时，你可以在 vim 后面用 - 或 -- 指定各种选项。

概述

大多数情况下，vim 用下面的命令启动后即可对文件 file 进行编辑：

```
$ vim file
```

但 vim 启动命令更一般的形式是下面这样：

```
$ vim [option | file] ..
```

选项 option 和文件名 file 可以混合使用任意多次，但选项如果带有参数时要小心。

注：本节讨论不涉及 Ex 模式。

编辑模式

对于如何开始编辑，vim 支持 5 种模式。

1) `file`

一或多个文件名。例如：

```
$ vim sailor.h
```

2) `-`

编辑缓冲区，由标准输入读取。例如：

```
$ ls *.c | vim -
```

用 `:write cfiles` 命令将缓冲区内容写到文件 `cfiles`

`cfiles` 中的内容：`./sailor_vim.c ./sailor.c`

3) `-t {tag}`

在 `tags` 中寻找标签 `tag`，编辑相关文件。例如：

```
$ ctags sailor.c sailor.h
```

```
$ vim -t linear
```

vim 跳转到 `sailor.c` 中 `linear()` 的定义。

4) `-q [errorfile]`

进入快速修复模式，即顺序加载编译器输出错误，跳转到错误行。

5) 空

不属于上面任何形式，开始编辑新的缓冲区。

启动选项

在 `vim` 启动过程中，可以指定不同选项，相邻的 `-` 后面的单字符选项可以合并在一起。`--` 是选项终止符，它后面不能再出现任何选项。

下面给出一些比较常见的 `vim` 启动选项。

-h

--help

-h

显示帮助信息，然后退出。例如：

```
$ vim -h
```

```
VIM - Vi IMproved 7.4 (2013 Aug 10, compiled Oct 13 2020 16:04:38)
```

```
usage: vim [arguments] [file ..]      edit specified file(s)
      or: vim [arguments] -            read text from stdin
      or: vim [arguments] -t tag        edit file where tag is defined
      or: vim [arguments] -q [errorfile] edit file with first error
```

Arguments:

```
--          Only file names after this
-v          Vi mode (like "vi")
-e          Ex mode (like "ex")
-E          Improved Ex mode
-s          Silent (batch) mode (only for "ex")
-d          Diff mode (like "vimdiff")
-y          Easy mode (like "evim", modeless)
-R          Readonly mode (like "view")
-Z          Restricted mode (like "rvim")
-m          Modifications (writing files) not allowed
-M          Modifications in text not allowed
-b          Binary mode
```

--version

--version

显示版本信息，然后退出。例如：

```
$ vim --version
```

```
VIM - Vi IMproved 7.4 (2013 Aug 10, compiled Oct 13 2020 16:04:38)
Included patches: 1-1089
Extra patches: 1.2.0050
Modified by pkg-vim-maintainers@lists.alioth.debian.org
Compiled by pkg-vim-maintainers@lists.alioth.debian.org
Huge version without GUI. Features included (+) or not (-):
+acl                +farsi              +mouse_netterm      +tag_binary
+arabic             +file_in_path       +mouse_sgr           +tag_old_static
+autocmd            +find_in_path       +mouse_sysmouse      +tag_any_white
+balloon_eval       +float              +mouse_urxvt         -tcl
-browse            +folding            +mouse_xterm         +terminfo
++builtin_terms     +footer             +multi_byte          +termresponse
+byte_offset        +fork()             +multi_lang          +textobjects
+channel            +gettext            -mzscheme            +timers
+cinindent          +hangul_input       +netbeans_intg       +title
+clientserver       +iconv              +packages            +toolbar
+clipboard          +insert_expand      +path_extra          +user_commands
+cmdline_compl      +job                +perl               +vertsplit
+cmdline_hist       +jumplist           +persistent_undo     +virtualedit
+cmdline_info       +keymap             +postscript          +visual
+comments           +langmap            +printer             +wiswlextra
+conceal            +libcall            +profile             +wininfo
+cryptv             +linebreak          -python              +wrapmode
+ecscope            +listcmds           +python3             +wildignore
+cursorbind         +localmap           +quickfix            +wildmenu
+cursorshape        +lua                +runtime             +windows
+dialog_con        +menu               +rightleft           +writebackup
+diff              +mksession          -ruby                -X11
+digraphs           +modify_fname       +scrollbind          +xfontset
-dnd               +mouse              +signs               +xim
+ebcdic            +mouseshape         +smartindent         +xmp
+emacs_tags         +mouse_dec          +startuptime         +xterm_clipboard
+eval              +mouse_gpm          +statusline          +xterm_save
+extra_search       +mouse_jsbterm      +syntax              -zip
   system vimrc file: "$VIM/vimrc"
     user vimrc file: "$HOME/.vimrc"
 2nd user vimrc file: "~/.vim/vimrc"
     user exrc file: "$HOME/.exrc"
 fall-back for $VIM: "/usr/share/vim"
Compilation: gcc -c -I. -Iproto -DHAVE_CONFIG_H -Wdate-time -g -O2 -fPIE -fstack-protector-strong -Wformat -Werror=format-security -U_FORTIFY_SOURCE -D_FORTIFY_SOURCE=1
Linking: gcc -Wl,-Bsymbolic-functions -fPIE -pie -Wl,-z,relro -Wl,-z,mov -Wl,-as-needed -o vim -lm -ltinfo -lnsl -lselinux -lacl -lattr -lgpm -ldl -L/usr/lib/python3.5/config-3.5m-x86_64-linux-gnu -lpthread3.5m -lpthread -ldl -lutil -ln
```

-y -n --noplugin

-y

进入 `easy` 模式。

打开 `vim` 后，自动进入编辑模式，就像“点击-输入”类型编辑器一样。

按下 `C-l` 进入命令模式。

-n

不使用交换文件，但也就无法从交换文件恢复崩溃的编辑会话了。

--noplugin

跳过加载插件。例如：

```
$ vim --noplugin
```

--starttime

--starttime {file}

记录启动过程中的时间消耗，写入文件 file。例如：

```
$ vim --starttime timingmsg sailor.c
```

```
[lettuce@lettuce:~/source/480-vim_base$ head -n 20 timingmsg
```

```
times in msec
clock self+sourced self: sourced script
clock elapsed:      other lines

000.003 000.003: --- VIM STARTING ---
000.092 000.089: Allocated generic buffers
000.162 000.070: locale set
000.166 000.004: window checked
000.664 000.498: inits 1
000.713 000.049: parsing arguments
000.715 000.002: expanding arguments
000.725 000.010: shell init
001.039 000.314: Termcap init
001.078 000.039: inits 2
001.229 000.151: init highlight
001.698 000.367 000.367: sourcing /usr/share/vim/vim74/debian.vim
002.395 000.398 000.398: sourcing /usr/share/vim/vim74/syntax/syncolor.vim
002.618 000.721 000.323: sourcing /usr/share/vim/vim74/syntax/synload.vim
```

+ [n]

+ [n]

打开 vim 后，跳转到编辑文件的第 n 行。例如：

```
$ vi +10 sailor.c
```

```
1 /*
2  * sailor.c is a simple C language source code file.
3  * This is a example source file, written with vim.
4  */
5
6 #include <stdio.h>
7
8 double linear(double a, double b, double x)
9 {
10     return a + b * x;
11 }
12
13 int main(int argc, char *argv[])
14 {
15     printf("3+2*1.5 = %.2f\n",
16           linear(3, 2, 1.5));
17     return 0;
18 }
19
20 // Edited by vim
~
```

`+/[pattern]`

`+/[pattern]`

打开 `vim` 后，跳转到编辑文件首次匹配模式 `pattern` 的行。如果这样的行不存在，则跳到最后一行。例如：

```
$ vi +/linear sailor.c
```

```
1 /*
2  * sailor.c is a simple C language source code file.
3  * This is a example source file, written with vim.
4  */
5
6 #include <stdio.h>
7
8 double linear(double a, double b, double x)
9 {
10     return a + b * x;
11 }
12
13 int main(int argc, char *argv[])
14 {
15     printf("3+2*1.5 = %.2f\n",
16           linear(3, 2, 1.5));
17     return 0;
18 }
19
20 // Edited by vim
~
```

-r -R

-r

进入恢复模式。如果 `-r` 后面没有参数，则列出已存在的交换文件；否则，读取交换文件并尝试恢复崩溃的编辑会话。例如：

```
$ vim -r sailor.h
```

```
Using swap file ".sailor.h.swp"
Original file "~/source/480-vim_base/sailor.h"
Recovery completed. You should check if everything is OK.
(You might want to write out this file under another name
and run diff with the original file to check for changes)
You may want to delete the .swp file now.
```

```
Press ENTER or type command to continue
```

-R

进入只读模式，相当于 `view` 命令。

你仍然可以修改缓冲区，但 `vim` 会阻止你错误地写入文件。如果你确实想要写入文件，可以使用强制写命令 `:w!`（后跟感叹号）。

如果想要取消只读模式，用 `:set noro` 命令将其取消。

-m -M

-m

不允许修改被写入文件。

可通过 `write` 选项重新打开。

-M

不允许修改。

关闭 `modifiable` 选项，同时关闭 `write` 选项。

可通过修改 `modifiable` 和 `write` 选项重新打开。

-Z -b -C -N

-Z

严格模式，禁止所有使用外部 `shell` 的命令。

-b

二进制模式，文件 I/O 只将 `<NL>` 认作行分隔符。

-C

兼容 `vi` 模式，其他配置文件中的设置可能将它覆盖。

如果该选项被覆盖，打开 `vim` 后，输入 `:verbose set compatible?`

可以查看是哪个文件最终设置了非兼容模式。例如：

```
nocompatible
```

```
    Last set from /usr/share/vim/vim74/debian.vim
```

-N

设置非兼容模式。

-l -A -F -H

-l

L i s p 模式。

-A

A r a b i c（阿拉伯语）模式。

-F

F a r s i（波斯语）模式。

-H

H e b r e w（希伯来语）模式。

-o -O -p

-o [N]

打开 N 个窗口，水平排列。

如果待编辑文件少于 N，剩余的打开空白窗口。

-O [N]

打开 N 个窗口，垂直排列。

如果待编辑文件少于 N，剩余的打开空白窗口。

-p [N]

打开 N 个标签页。

标签页的最大数量由 `tabpagemax` 选项设定（默认 10）。

-T -d -u -i

-T {terminal}

设置终端类型为 `terminal`。

通常没必要这样做，因为 `vim` 会自动识别你使用的终端。

-d

开启比较模式，类似 `vimdiff`。例如：

```
$ vim -d sailor.h sailor.h.old
```

-u {vimrc}

加载 `vim` 配置文件 `vimrc` 进行初始化。

如果 `vimrc` 是 `NONE`，则跳过来自文件和环境变量的初始化以及插件的加载；

`NORC` 和 `NONE` 一样，但不会跳过加载插件。

-i {viminfo}

使用 `viminfo`，而不是默认的 `viminfo` 文件。

如果 `viminfo` 是 `NONE`，则不会读写 `viminfo` 文件。

-w -W

-w {scriptout}

记录你的所有键入，保存到文件 `scriptout`。

如果 `scriptout` 文件已存在，则在文件尾追加记录。

例如：

```
$ vim -W script.rec sailor.h
```

打开文件后：

=====

按下 `G` 来到最后一行，按下 `o` 追加一行，回车，输入：

```
// Edited by vim
```

输入 `:wq` 命令保存文件并退出 `vim`。

=====

以上按键输入均记录在文件 `script.rec`，内容如下：

```
Go^M// Edited by vim^[:wq^M
```

-W {scriptout}

同 `-w`，但当 `scriptout` 文件存在时，覆盖原内容。

-S

`-s {scriptin}`

读取记录按键输入的文件 `scriptin`，应用到编辑文件。

如果在编辑会话退出之前已读到了文件末尾，则等待用户输入。

例如：

```
$ vim -s script.rec sailor.c
```

`vim` 读取 `script.rec`，将里面的按键输入应用到 `sailor.c`。因此，`sailor.c` 的最后 3 行的内容变为：

```
-----  
}
```

```
// Edited by vim
```

```
-----
```