



Linux操作系统

授课教师：刘二小 副教授

单位：杭州电子科技大学

通信工程学院



第八章 编辑器及gcc编译器

- 编辑器是所有计算机系统中常用的一种工具，用户在使用计算机的时候，往往需要自己建立文件，编写程序，这些工作都需要编辑器，这里我们介绍在Linux下常用的三种编辑器，分别是最基本基于字符界面的vi和nano，基于图形界面的gedit。
- gcc（GNU Compiler Collection，GNU编译器套装），是一套由GNU开发的编程语言**编译器**，自由软件，GNU计划的关键部分，类Unix系统及苹果电脑Mac OS X 操作系统的标准编译器，被认为是跨平台编译器的事实标准，可处理多种语言。



8.1 三种编辑器

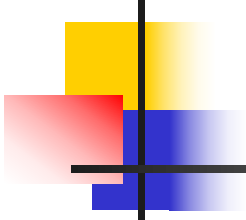
•8.1.1 vi编辑器

vi是visual interface的简称，是Linux中最常用的编辑器，**vim是它的改进版本**。

它的文本编辑功能十分强大，但使用起来比较复杂。初学者可能感到困难，

vi是一种模式编辑器，不同的按钮和键击可以更改不同的“模式”，可以在“状态条”显示当前模式。

命令	功能
vi filename	打开或新建文件，并将光标置于第一行首
vi +n filename	打开文件，并将光标置于第 n 行首
vi + filename	打开文件，并将光标置于最后一行首
vi + /str filename	打开文件，并将光标置于第一个与 str 匹配的串处
vi -r filename	在上次使用 vi 编辑时系统崩溃，恢复 filename
vi filename1...filenamen	打开多个文件，依次编辑



2. 三种工作模式

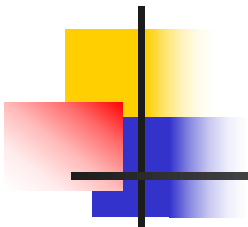
vi有三种工作模式：命令行模式、输入模式、末行模式。

(1) 命令行模式

当进入vi时，它处在命令模式。在这种模式下，用户可通过vi命令对文件的内容进行处理，比如删除、移动、复制等。

例如：vi 文件名 此时进入命令行模式。

在这个模式中，用户可以输入各种合法的vi命令，管理自己的文档。从键盘上输入的任何字符都被当做编辑命令，如果输入的字符是合法的vi命令，则vi接受用户命令并完成相应的动作。在命令模式下输入命令切换到输入模式，在输入模式下，若要用命令，按下ESC键，返回命令模式，再输入命令。



(2) 输入模式

在输入模式下，用户能在光标处输入内容，或通过光标键移动光标。也可通过按ESC键返回命令模式。

命令行进入输入模式的按键如下表所示：

命令	功能
i	从目前光标所在处插入
I	从目前所在行的第一个非空格符处开始插入
a	从目前光标所在的下一个字符处开始插入
A	从光标所在行的最后一个字符处开始插入
o	从目前光标所在行的下一行处插入新的一行
O	从目前光标所在处的上一行插入新的一行
r	替换光标所在的那一个字符一次
R	替换光标所在处的文字，直到按下 Esc 键为止



(3) 末行模式

- 命令行模式下按 “:” 键进入末行模式，提示符为 “:”。
- 末行命令执行后，vi自动回到命令模式。若在末行模式的输入过程中，可按退格键将输入的命令全部删除，再按一下退格键，即可回到命令模式。
- 末行模式的可用按键及含义如表所示：

:w	将编辑的数据保存到文件中（不退出）
:w!	若文件属性为“只读”时，强制写入该文件
:q	退出vi
:q!	强制退出不保存文件
:wq	保存后退出vi
:w filename	将编辑的数据保存成另一个文件（另存为）
: /word	向下寻找一个名称为word的字符串
: ?word	向上寻找一个名称为word的字符串
:n1, n2s/word1/word2/g 加c, 按y来确认修改。	在第n1与n2行之间寻找word1字符串，并替换为word2 (用word2替换word1)
:1, \$s/word1/word2/g	全文查找word1字符串，并将该它替换为word2

★vi编辑器的三种工作模式之间的转换:

命令行模式→输入模式: i, l, a, A,.....

输入模式→命令行模式: Esc

命令行模式→末行模式: :

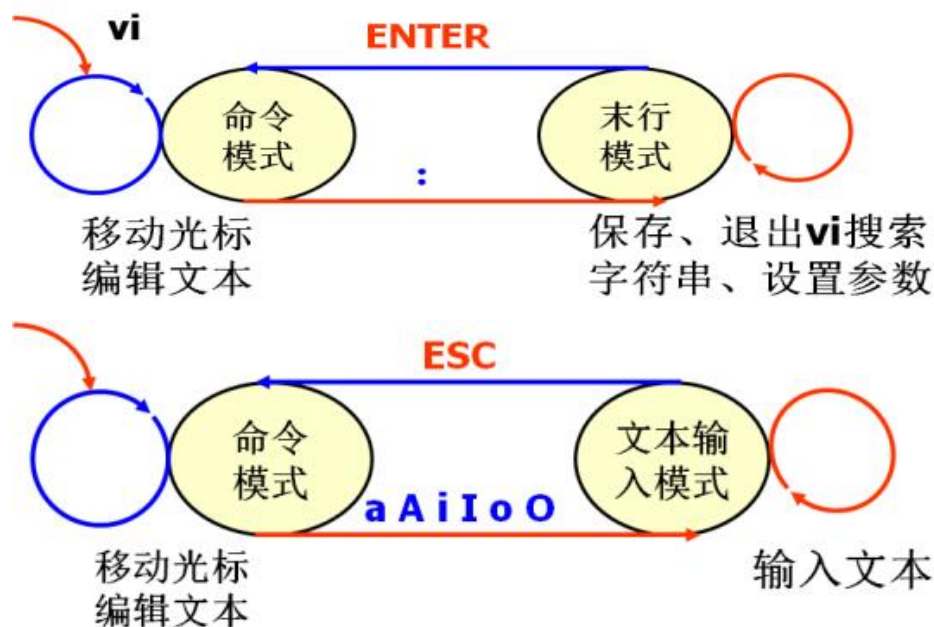
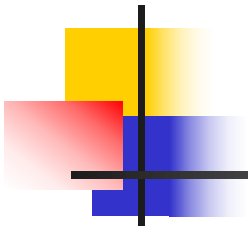


图8.1 模式转换示意图



3.光标操作命令：**命令行模式下**，移动光标的方法如表所示：

h或向左箭头键	光标向左移动一个字符
j或向下箭头键	光标向下移动一个字符
k或向上箭头键	光标向上移动一个字符
l或向右箭头键	光标向右移动一个字符
+	光标移动到非空格符的下一行
-	光标移动到非空格符的上一行
n<space>	按下数字n后再按空格键，光标会向右移字符
0或功能键Home	移动到这一行的行首
\$或功能键End	移动到这一行的行尾
H	光标移动到屏幕的第一行的第一个字符
M	光标移动到屏幕的中央的那一行的第一个字符
L	光标移动到屏幕的最后一行的第一个字符
G	光标移动到这个文件的最后一行
nG	n为数字。移动到这个文件的第n行
gg	移动到这个文件的第一行。相当于1G
n[Enter]	n为数字。光标向下移动n行



4. 屏幕操作命令

- 在**命令行模式**和**输入模式**下都可以使用屏幕滚动命令，以屏幕为单位的光标操作，常用于滚屏和分页的按键如表所示：

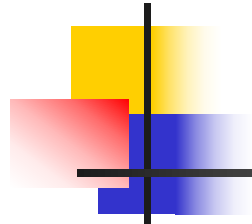
按键	功能
[Ctrl]+f	屏幕向下移动一页，相当于[Page Down]按键
[Ctrl]+b	屏幕向上移动一页，相当于[Page Up]按键
[Ctrl]+d	屏幕向下移动半页
[Ctrl]+u	屏幕向上移动半页



5. 文本修改命令

• **在命令行模式**，可以对文本进行修改，包括对文本内容的删除、复制、粘贴等的操作。

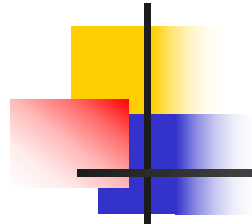
按键	功能
x	删除光标所在位置上的字符
dd	删除光标所在行
n+x	向后删除n个字符，包含光标所在位置
n+dd	向下删除n行内容，包含光标所在行
yy	将光标所在行复制
n+yy	将从光标所在行起向下的n行复制
n+yw	将从光标所在位置起向后的n个字符串(单词)复制
p	将复制(或最近一次删除)的字符串(或行)粘贴在当前光标所在位置
u	撤销上一步操作
.	重复上一步操作



(1)复制/etc/manpath.config文件到当前目录，使用vi打开该目录下的manpath.config文件，如图所示：

```
lex@lex-virtual-machine:~$ cp /etc/manpath.config ./
lex@lex-virtual-machine:~$ ls
20080808      公共的  图片  音乐  lesson6_test  tmp
20080808.tar  模板  文档  桌面  manpath.config
20080808.tar.gz 视频  下载  lesson  snap
```

```
# manpath.config
#
# This file is used by the man-db package to configure the man-
# cat paths.
# It is also used to provide a manpath for those without one by
# aining
# their PATH environment variable. For details see the manpath(
# man page.
#
# Lines beginning with '#' are comments and are ignored. Any co
# nation of
# tabs or spaces may be used as 'whitespace' separators.
#
# There are three mappings allowed in this file:
# -----
# MANDATORY_MANPATH      manpath_element
# MANPATH_MAP             path_element  manpath_element
# MANDB_MAP               global_manpath [relative_catpath]
# -----
# every automatically generated MANPATH includes these fields
#
#MANDATORY_MANPATH      /usr/src/pvm3/man
#
MANDATORY_MANPATH      /usr/man
MANDATORY_MANPATH      /usr/share/man
MANDATORY_MANPATH      /usr/local/share/man
#-----
```

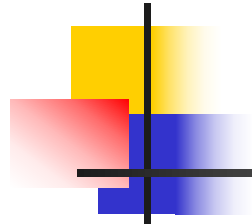


(2)在vi中设置一下行号。在末行模式输入：set nu （设置行号）

```
1 # manpath.config
2 #
3 # This file is used by the man-db package to configure the man
   and cat paths.
4 # It is also used to provide a manpath for those without one l
   y examining
5 # their PATH environment variable. For details see the manpat
   (5) man page.
6 #
7 # Lines beginning with '#' are comments and are ignored. Any c
   ombination of
8 # tabs or spaces may be used as 'whitespace' separators.
9 #
10 # There are three mappings allowed in this file:
11 # -----
12 # MANDATORY_MANPATH          manpath_element
13 # MANPATH_MAP                path_element  manpath_element
14 # MANDB_MAP                  global_manpath [relative_catpath]
15 #-----
16 # every automatically generated MANPATH includes these fields
17 #
18 #MANDATORY_MANPATH          /usr/src/pvm3/man
19 #
20 MANDATORY_MANPATH          /usr/man
21 MANDATORY_MANPATH          /usr/share/man
22 MANDATORY_MANPATH          /usr/local/share/man
23 #-----
:set nu                                1,1      顶端
```

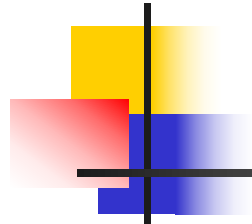
(3)移动到第一行，并且向下查找字符串DB_MAP。

```
lex@lex-virtual-machine: ~  
5 # their PATH environment variable. For details see the manpath  
  (5) man page.  
6 #  
7 # Lines beginning with '#' are comments and are ignored. Any c  
  ombination of  
8 # tabs or spaces may be used as 'whitespace' separators.  
9 #  
10 # There are three mappings allowed in this file:  
11 # -----  
12 # MANDATORY_MANPATH          manpath_element  
13 # MANPATH_MAP                path_element  manpath_element  
14 # MANDB_MAP                  global_manpath [relative_catpath]  
15 # -----  
16 # every automatically generated MANPATH includes these fields  
17 #  
18 #MANDATORY_MANPATH          /usr/src/pvm3/man  
19 #  
20 MANDATORY_MANPATH          /usr/man  
21 MANDATORY_MANPATH          /usr/share/man  
22 MANDATORY_MANPATH          /usr/local/share/man  
23 # -----  
24 # set up PATH to MANPATH mapping  
25 # ie. what man tree holds man pages for what binary directory.  
26 #  
27 #          *PATH*          ->          *MANPATH*  
28 #  
29 MANPATH_MAP          /bin          /usr/share/man
```

(4)将第66到第71行之间的man修改为MAN，并且一个一个提示是否需要修改。
在末行模式输入：66，71s/man/MAN/gc，之后按y来确认修改

```
lex@lex-virtual-machine: ~  
57 # location of catpaths and the creation of database caches; it  
    has no effect  
58 # on privileges.  
59 #  
60 # Any manpaths that are subdirectories of other manpaths must  
    be mentioned  
61 # *before* the containing manpath. E.g. /usr/man/preformat mus  
    t be listed  
62 # before /usr/man.  
63 #  
64 #          *MANPATH*          ->          *CATPATH*  
65 #  
66 MANDB_MAP          /usr/MAN          /var/cache/MAN/fsstnd  
67 MANDB_MAP          /usr/share/MAN          /var/cache/MAN  
68 MANDB_MAP          /usr/local/MAN          /var/cache/MAN/oldloca  
    l  
69 MANDB_MAP          /usr/local/share/MAN          /var/cache/MAN/local  
70 MANDB_MAP          /usr/X11R6/MAN          /var/cache/MAN/X11R6  
71 MANDB_MAP          /opt/MAN          /var/cache/MAN/opt  
72 MANDB_MAP          /snap/man          /var/cache/man/snap  
73 #
```



(5)在末行模式下输入u恢复到上一步状态，如图所示，MAN 恢复到小写状态。

```
5 MANDB_MAP      /usr/man          /var/cache/man/fsstnd
6 MANDB_MAP      /usr/share/man    /var/cache/man
7 MANDB_MAP      /usr/local/man  /var/cache/man/oldlocal
8
9 MANDB_MAP      /usr/local/share/man  /var/cache/man/local
10 MANDB_MAP      /usr/X11R6/man    /var/cache/man/X11R6
11 MANDB_MAP      /opt/man          /var/cache/man/opt
12 MANDB_MAP      /snap/man         /var/cache/man/snap
13 #
```



n+yy

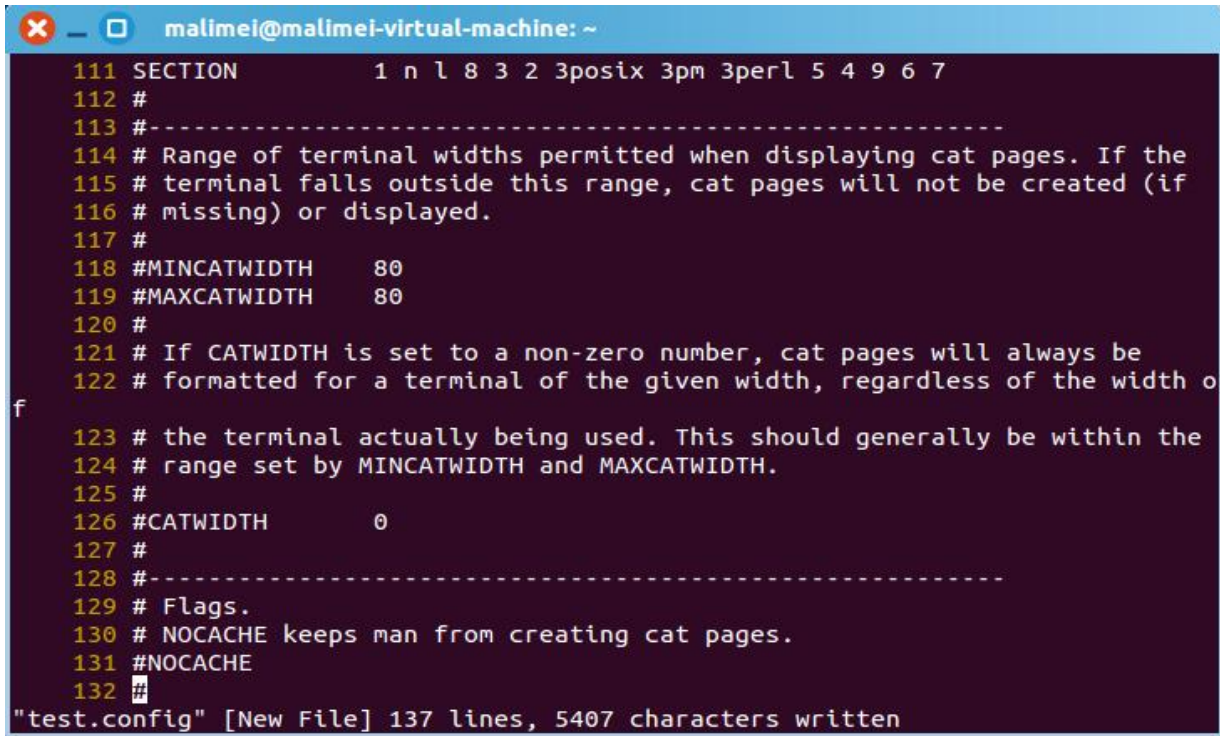
将从光标所在行起向下的n行复制

(6)复制65到第70行之间的内容，并且粘贴到最后一行之前。按ESC键转到命令行模式输入65G，然后按下6yy，最后一行会出现复制六行的说明字样，如图所示，按下G到最后一行，再按p粘贴六行，如图所示。

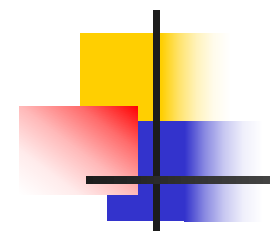
```
63 #
64 #          *MANPATH*      ->      *CATPATH*
65 #
66 MANDB_MAP      /usr/man          /var/cache/m
67 d
68 MANDB_MAP      /usr/share/man     /var/cache/m
69 MANDB_MAP      /usr/local/man     /var/cache/m
70 cal
71 MANDB_MAP      /usr/local/share/man /var/cache/m
72 MANDB_MAP      /usr/X11R6/man     /var/cache/m
73 #
74 #-----
75 # Program definitions.  These are commented out by d
76 s the value
6 lines yanked
```

```
129 #-----
130 # Flags.
131 # NOCACHE keeps man from creating cat pages.
132 #NOCACHE
133 #
134 MANDB_MAP      /usr/man          /var/cache/man/fsstr
135 d
136 MANDB_MAP      /usr/share/man     /var/cache/man
137 MANDB_MAP      /usr/local/man     /var/cache/man/oldl
138 cal
6 more lines
```


(7)将此文件另存为test.config。在末行模式输入：w test.config，如图所示。



```
malimei@malimei-virtual-machine: ~
111 SECTION      1 n l 8 3 2 3posix 3pm 3perl 5 4 9 6 7
112 #
113 #-----
114 # Range of terminal widths permitted when displaying cat pages. If the
115 # terminal falls outside this range, cat pages will not be created (if
116 # missing) or displayed.
117 #
118 #MINCATWIDTH    80
119 #MAXCATWIDTH    80
120 #
121 # If CATWIDTH is set to a non-zero number, cat pages will always be
122 # formatted for a terminal of the given width, regardless of the width o
123 # the terminal actually being used. This should generally be within the
124 # range set by MINCATWIDTH and MAXCATWIDTH.
125 #
126 #CATWIDTH       0
127 #
128 #-----
129 # Flags.
130 # NOCACHE keeps man from creating cat pages.
131 #NOCACHE
132
"test.config" [New File] 137 lines, 5407 characters written
```

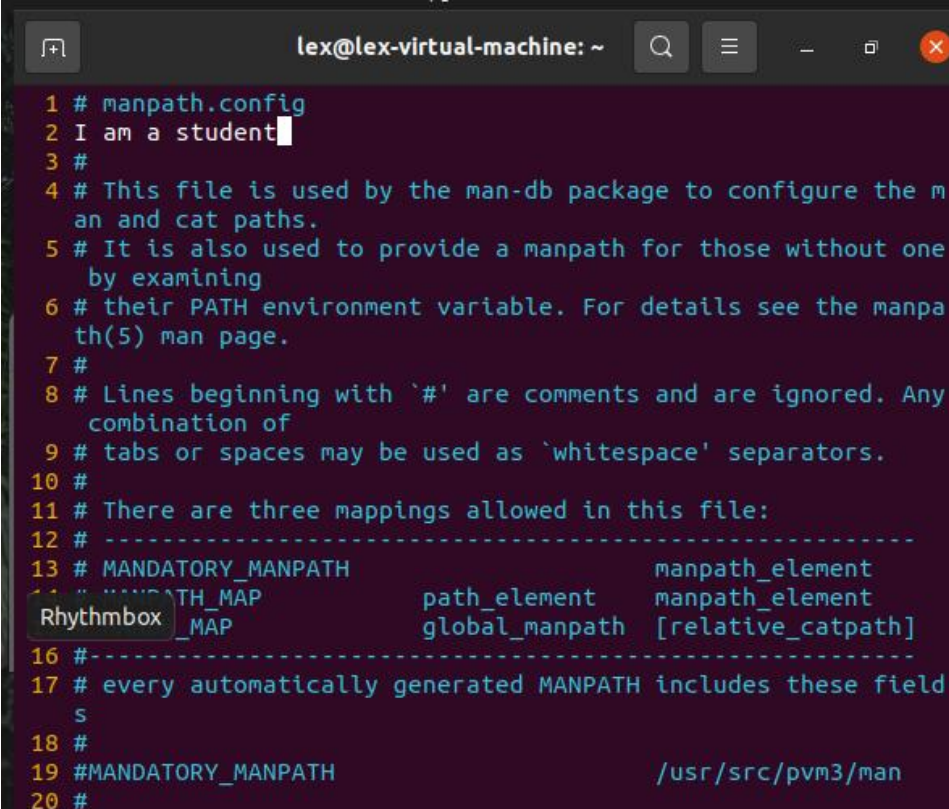


(8)在第74行，删除58个字符。在命令行先输入74G，再按下58x

```
72 MANDB_MAP /snap/man /var/cache/man/sn
73 #
74 
75 # Program definitions. These are commented out by default
76 # of the definition is already the default. To change: u
77 # definition and modify it.
78 #
79 #DEFINE pager pager
80 #DEFINE cat cat
81 #DEFINE tr tr '\255\267\264\327' '\055\157\0
170'
82 #DEFINE grep grep
83 #DEFINE troff groff -mandoc
```

74,0-1 5

(9)在第一行新增一行，并输入I am a student。在命令行先按下1G，再按下O来新增一行转到插入模式，输入I am a student，如图8.17所示，按Esc键退出输入模式，按下：wq，保存文件。



```
lex@lex-virtual-machine: ~
1 # manpath.config
2 I am a student
3 #
4 # This file is used by the man-db package to configure the m
  an and cat paths.
5 # It is also used to provide a manpath for those without one
  by examining
6 # their PATH environment variable. For details see the manpa
  th(5) man page.
7 #
8 # Lines beginning with '#' are comments and are ignored. Any
  combination of
9 # tabs or spaces may be used as 'whitespace' separators.
10 #
11 # There are three mappings allowed in this file:
12 # -----
13 # MANDATORY_MANPATH          manpath_element
14 # MANDATORY_MAP              path_element  manpath_element
15 Rhythmbox _MAP               global_manpath [relative_catpath]
16 #-----
17 # every automatically generated MANPATH includes these field
  s
18 #
19 #MANDATORY_MANPATH          /usr/src/pvm3/man
20 #
```



6.其他命令

(1)块选择

■ 选择一行或多行，可以使用如表所示的块选择按键。

按键	功能
V	行选择，将光标经过的地方反白选择
[Ctrl]+v	块选择，可以用长方形的方式选择数据

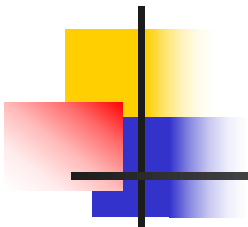
(2)多文件编辑

■ 多文件编辑常用键如表所示：

按键	功能
:n	编辑下一个文件
:N	编辑上一个文件
:files	列出目前这个vim打开的所有文件

```
LEX:x:1006:1006::/home/LEX/:/bin/bash
:files
 1 %a    "passwd"
 2 #     "shadow"
请按 ENTER 或其它命令继续
```

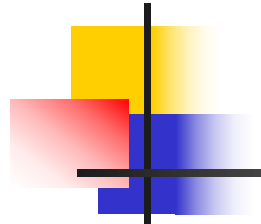
第 53 行
第 54 行



(3)多窗口功能

打开多个窗口，编辑多个文件，光标可在不同窗口间切换，常用的按键如表所示。

:sp filename	打开一个新窗口，如filename，表示在新窗口新打开一个新文件，否则表示两个窗口为一个文件内容
[Ctrl]+w+j	先按下Ctrl不放，再按下w后放开所有的按键，然后再按下j，则光标可移动到下方的窗口
[Ctrl]+w+k	同上，不过光标移动到上面的窗口
[Ctrl]+w+q	结束离开当前窗口



vi /etc/passwd 先打开一个文件，用命令 (sp)调入另外一个文件/etc/shadow，同时显示两个文件, Ctrl+w+j 移到下方窗口，[Ctrl]+w+k 光标移动到上面的窗口。

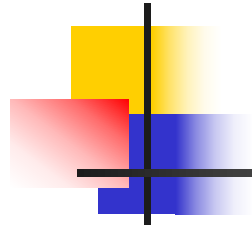
终端

root@malimei-virtual-machine: /

kernoops:x:106:65534:Kernel Oops Tracking Daemon,,,:/bin/false
rtkit:x:107:114:RealtimeKit,,,:/proc/bin/false
saned:x:108:115:/home/saned/bin/false
whoopsie:x:109:116:/nonexistent/bin/false
speech-dispatcher:x:110:29:Speech Dispatcher,,,:/var/run/speech-dispatcher/bin/sh
avahi:x:111:117:Avahi mDNS daemon,,,:/var/run/avahi-daemon/bin/false
lightdm:x:112:118:Light Display Manager:/var/lib/lightdm/bin/false
colord:x:113:121:colord colour management daemon,,,:/var/lib/colord/bin/false
hplip:x:114:7:HPLIP system user,,,:/var/run/hplip/bin/false
pulse:x:115:127:PulseAudio daemon,,,:/var/run/pulse/bin/false
malimei:x:1000:1000:malimei,,,:/home/malimei/bin/bash
gdm:x:116:125:Gnome Display Manager:/var/lib/gdm/bin/false
sshd:x:117:65534:/var/run/sshd:/usr/sbin/nologin
xiaoli:x:1001:1001,,,:/home/xiaoli/bin/bash
xiaowang:x:1002:1002:/home/xiaowang
xiao:x:1005:1005,,,:/home/user2/bin/bash
ooo:x:1000:1000,,,:/home/ooo/bin/bash
bb:x:1009:1009:/home/bb
kk:x:1010:1010:/home/kk
jj:x:1012:1012:/home/jj
jj2:x:1013:1013:/home/jj2
kk22:x:1014:1014:/home/kk22
:sp /etc/shadow

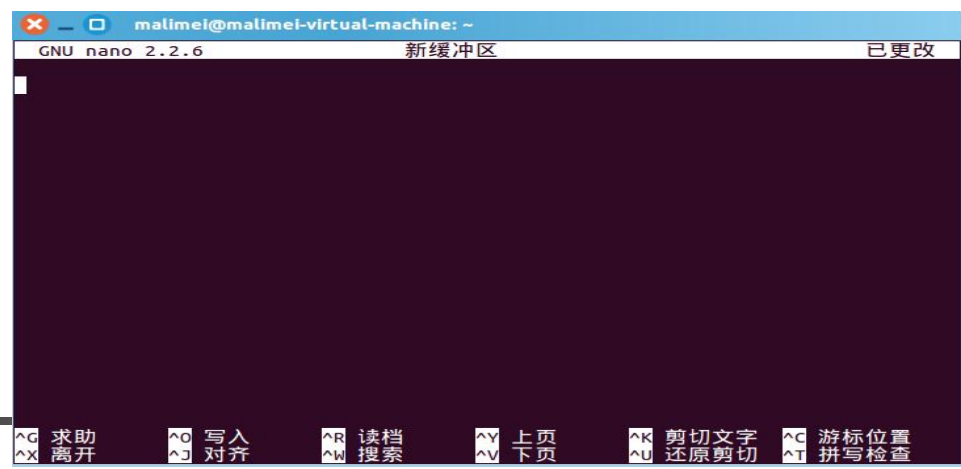
root@lex-virtual-machine: /etc

lex2:\$6\$0mTol2thYdr307Dy\$wzYIYxIeOruGNtnfuxp7GpABSNp5ZPgKOHjhdET
RtFmUYbCoZrvjcXjinhgFHA0zp0.Fe.NMfdkgCoQ9qFzKW1:18730:0:99999:7:
:
lex3:::18734:0:99999:7:::
lex4:\$6\$zyFzMd1hdMsCY7g4\$zDQrLrVPwbT8/00JvLCZ4QyyquaNFLnWYS4Hit9
/bHsCJfKgGtKsl9jDtUyTRhhZSILNfYmOxQn7Z2V71z34s0:18734:0:99999:7:
:
lex5:\$6\$712v2WVck6BHjuxN\$SFIm0ltewX43uj3fSyq6XXFXSaK30hiMNR7l5k
NRPqm9PuUNmrf/LlfyNHTeDEHUBzOCKNrLacIpFqYHvwIV/:18734:0:99999:7:
:
LEX:::18737:0:1:7:::
~
~
/etc/shadow 54,5 底端
pulse:x:123:129:PulseAudio daemon,,,:/var/run/pulse:/usr/sbin/nologin
hplip:x:124:7:HPLIP system user,,,:/run/hplip/bin/false
gnome-initial-setup:x:125:65534:/run/gnome-initial-setup:/bin/false
gdm:x:126:131:Gnome Display Manager:/var/lib/gdm3/bin/false
lex:x:1000:1000:lex,,,:/home/lex/bin/bash
systemd-coredump:x:999:999:systemd Core Dumper:/usr/sbin/nologin
sshd:x:127:65534:/run/sshd:/usr/sbin/nologin
lex1:x:1001:1001,,,:/home/lex1/bin/bash
lex2:x:1002:1002,,,:/home/lex2/bin/bash



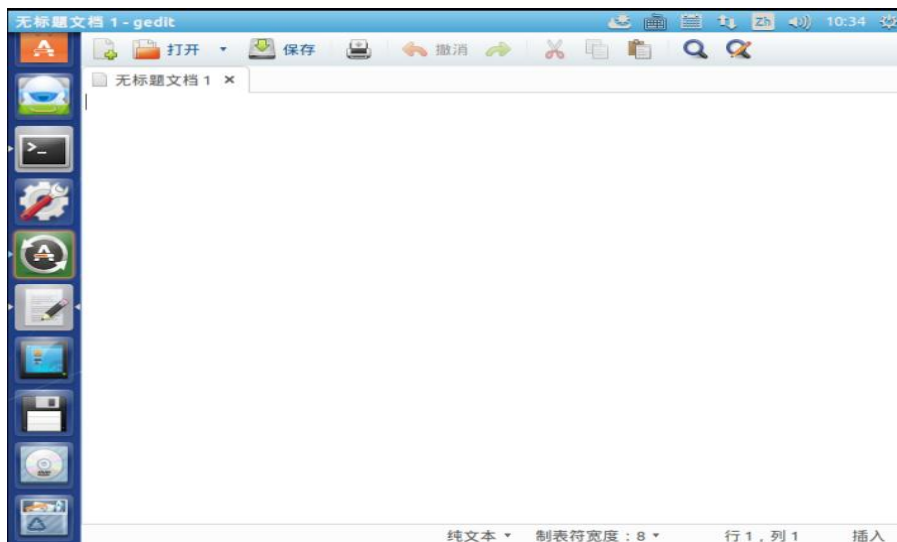
8.1.2 nano编辑器

- nano是Unix和类Unix系统中的一个轻量级文本编辑器，它比vi/vim要简单得多，比较适合Linux初学者使用。某些Linux发行版的默认编辑器就是nano，nano是遵守GNU通用公共许可证的自由软件，使用方便，在任何一个终端中键入nano命令即可打开nano编辑器。
- 在屏幕的下面显示功能键的使用，如：^k就表示ctrl+k 剪切当前行，将其内容保存到剪切板中，^U将剪切板的内容写入当前行。



8.1.3 gedit编辑器

- gedit是Linux桌面上一款小巧的文本编辑器，外观简单，仅在工具栏上具有一些图标及一排基本菜单，因是图形界面，所示使用方便。Gedit的启动方式：
- 打开终端，输入命令gedit，回车





8.2 gcc编译器

GCC (GNU Compiler Collection, GNU编译器套件), 是由 GNU 开发的编程语言编译器。它是以GPL许可证所发行的自由软件, 也是 GNU计划的关键部分。GCC原本作为 GNU操作系统的官方编译器, 现已被大多数类Unix操作系统 (如Linux、BSD、Mac OS X等) 采纳为标准的编译器, GCC同样适用于微软的Windows。GCC是自由软件过程发展中的著名例子, 由自由软件基金会以GPL协议发布。

GCC 原名为 GNU C 语言编译器 (GNU C Compiler), 因为它原本只能处理 C语言。GCC 很快地扩展, 变得可处理 C++。后来又扩展能够支持更多编程语言, 如Fortran、Pascal、Objective-C、Java、Ada、Go以及各类处理器架构上的汇编语言等, 所以改名 GNU编译器套件 (GNU Compiler Collection) 。

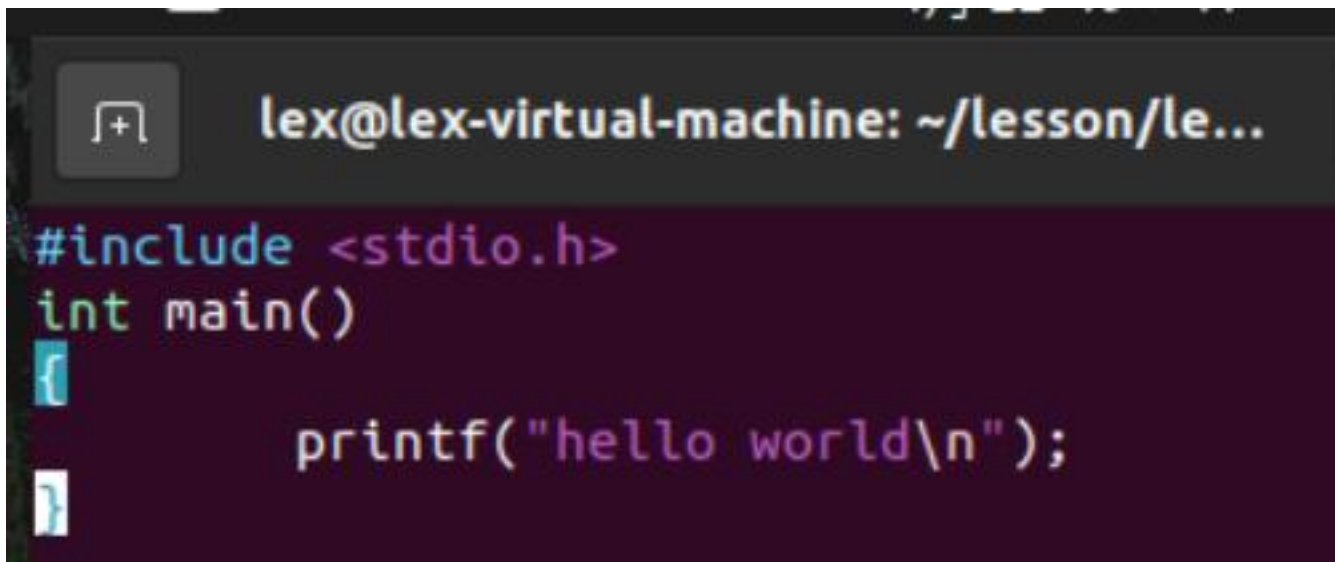
8.2.1 gcc编译器的使用

1.gcc编译流程

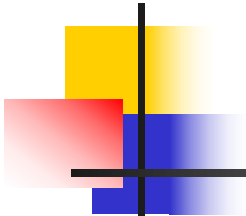
gcc的编译流程为预编译/预处理、编译、汇编(生成目标文件)、链接(生成可执行的文件)。

例：编译当前目录下的hello.c文件并执行。

(1)创建hello.c文件，并输入代码。



```
lex@lex-virtual-machine: ~/lesson/le...  
#include <stdio.h>  
int main()  
{  
    printf("hello world\n");  
}
```



(2)保存并退出后查看并执行hello.c，如图所示。

```
lex@lex-virtual-machine:~/lesson/lesson7$ cat hello.c
#include <stdio.h>
int main()
{
    printf("hello world\n");
}
lex@lex-virtual-machine:~/lesson/lesson7$ gcc hello.c -o hello.out
lex@lex-virtual-machine:~/lesson/lesson7$ ls
hello.c  hello.out
lex@lex-virtual-machine:~/lesson/lesson7$ ./hello.out
hello world
```

选项	含义
-c	生成目标
-S	编译后生成汇编文件
-E	预处理后即停止，不进行编译、汇编及连接
-g	在可执行文件中包含调试信息
-o file	指定输出文件file

注意:执行可执行文件方式: (1) **./**hello.out (2) /home/lex/lesson/lesson7/hello.out

8.2.2 gcc总体选项实例

程序编译4个过程:

预处理

- 主要处理C语言源文件中的#ifdef、#include、以及#define等命令,即将include文件插入到源文件中、将宏定义展开等;
- 输入*.c,生成中间文件*.i;

编译

- 检查语法,生成汇编,即把“C/C++”代码“翻译”成汇编代码;
- 输入的是中间文件*.i,编译后生成的是汇编语言文件*.s

汇编

- 汇编是将输入的汇编语言文件转换为目标代码(机器代码),可以使用-c选项完成

链接

- 将生成的目标文件与其他目标文件连接成可执行的二进制代码文件

*.c

预处理

*.i

编译

*.s

汇编

*.o

链接

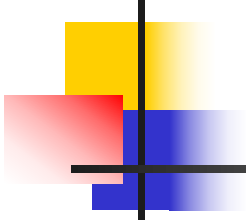
.(exe)



8.2.3 优化选项实例

一般来说，优化级别越高，生成可执行文件的运行速度也越快，但编译的时间就越长，因此在开发的时候最好不要使用优化选项，只有到软件发行或开发结束的时候才考虑对最终生成的代码进行优化。

选项	含义
-O0	不进行优化处理
-O1	基本的优化，使程序执行得更快
-O2	完成-O1级别的优化外，还要一些额外的调整工作，如处理器指令调度
-O4	开启所有优化选项
-Os	生产最小的可执行文件，主要用于嵌入式领域



例：比较gcc优化项的效果。

源程序example.c的代码如下：

```
#include<stdio.h>
```

```
int main()
```

```
{
```

```
    int x;
```

```
    int sum=0;
```

```
    for(x=1;x<1e9;x++)
```

```
    {
```

```
        sum=sum+x;
```

```
    }
```

```
    printf("sum=%d\n",sum);
```

```
}
```

```
lex@lex-virtual-machine:~/lesson/lesson7$ vi example.c
lex@lex-virtual-machine:~/lesson/lesson7$ gcc example.c -o example
lex@lex-virtual-machine:~/lesson/lesson7$ time ./example
sum=887459712

real    0m0.312s
user    0m0.308s
sys     0m0.004s
```

在编译源程序example.c过程中，不加任何优化选项，使用time命令查看程序执行时间。



其中time命令的输出结果由以下3部分组成:

- real: 程序的总执行时间, 包括进程的调度、切换等时间
- user: 用户进行执行的时间。
- sys: 内核执行的时间。

加入优化选项后使用time命令查看程序执行时间。

```
lex@lex-virtual-machine:~/lesson/lesson7$ gcc example.c -o example
lex@lex-virtual-machine:~/lesson/lesson7$ time ./example
sum=-1243309312

real    0m3.144s
user    0m3.134s
sys     0m0.000s
lex@lex-virtual-machine:~/lesson/lesson7$ gcc -O2 example.c -o example
lex@lex-virtual-machine:~/lesson/lesson7$ time ./example
sum=-1243309312

real    0m0.379s
user    0m0.374s
sys     0m0.004s
```

执行时间明显减少

注意:优化后个别情况执行时间会增加,与gcc内部优化时函数调用等很多因素有关!

8.2.4 加参数显示警告和出错信息

在编译过程中，编译器的报错和警告信息对于程序员来说是非常重要的信息。

例：编译example2.c程序，同时开启警告信息。

源程序example2.c的代码如下：

```
#include<stdio.h>
```

```
void main()
```

```
{
```

```
    int x; int sum=0;
```

```
    for(x=1;x<1e8;x++)
```

```
    {
```

```
        sum=sum+x;
```

```
    }
```

```
    printf("sum=%d\n",sum);
```

```
}
```

```
lex@lex-virtual-machine:~/lesson/lesson7$ gcc -Wall example2.c -o ex2
example2.c:2:6: warning: return type of 'main' is not 'int' [-Wmain]
  2 | void main()
    |         ^
example2.c: In function 'main':
example2.c:5:15: error: expected ';' before ':' token
  5 |     for(x=1;x<1e8:x++)
    |                   ^
    |                   ;
```




8.2.5 gdb调试器

- 在Unix/Linux系统中，调试工具为gdb(GNU symbolic debugger)。通过调试可以找到程序中的bug，它使用户能在程序运行时观察**程序的内部结构**和**内存的使用情况**。
- Windows下的各类IDE,如VS、VC、Dev-C++，自带了相应的调试器，Linux下利用GDB来实现。

1.gdb功能介绍

调试器使用：

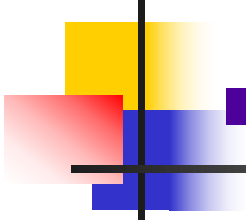
- 监视或修改程序中变量的值。
- 设置断点，以使程序在指定的代码行上暂停执行。
- 单步执行或程序跟踪。

gdb命令可以缩写，如list可缩写为l，kill可缩写为k，step可缩写为s等等。同样，在不引起歧义的情况下，可以使用tab命令进行自动补齐或查找某一类字符开始的命令。



■ gdb调试时的常用命令

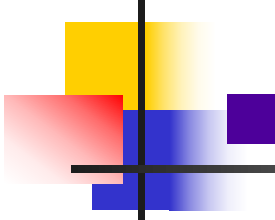
选项	功能
b reak	在代码里设置断点
c ontinue	继续break后的执行
bt	反向跟踪，显示程序堆栈
file	装入想要调试的可执行文件
k ill	终止正在调试的程序
l ist	列出产生执行文件的源代码的一部分
n ext	执行一行源代码，但不进入函数内部
s tep	执行一行源代码且进入函数内部
r un	执行当前被调试的程序
watch	监视一个变量的值，而不管它何时改变
set	设置变量的值
shell	在gdb内执行shell命令
p rint	显示变量或表达式的值
q uit	终止gdb调试
make	不退出gdb的情况下，重新产生可执行文件
where	显示程序当前的调用栈



■ gdb调试时的常用命令

➤ 增加断点break

格式	说明
break <函数名>	对当前正在执行的文件中的指定函数设置断点
break <行号>	对当前正在执行的文件中的特定行设置断点
break <文件名:行号>	对指定文件的指定行设置断点，最常用的设置断点方式
break <文件名:函数名>	对指定文件的指定函数设置断点
break <+/-偏移量>	当前指令行+/-偏移量处设置断点
break <*地址>	指定地址处设置断点



gdb调试时的常用命令

➤ 删除断点

(1) **delete** 命令删除断点和监视点，简写为 **d**。格式为 **delete <断点编号>**

(2) **clear** 命令删除已定义的断点。

可用命令包括：

clear <函数名>

clear <行号>

clear <文件名:行号>

clear <文件名:函数名>

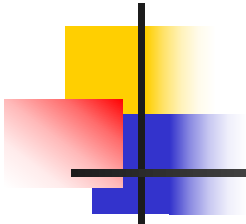
(3) **disable** 命令禁用断点。命令格式如下：

disable：禁用所有断点。

disable <断点编号>：禁用指定断点。

disable display <显示编号>：禁用 **display** 命令定义的自动显示。

disable mem <内存显示>：禁用 **mem** 命令定义的内存区域。

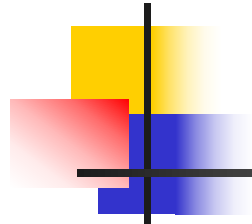


2. gdb的调试实例

以file.c程序为例，介绍Linux系统内程序调试的基本方法。

源程序file.c的代码如下：

```
#include <stdio.h>
static char buff[256];
static char* string;
int main()
{
    printf("please input a string:");
    gets(string);
    printf("\nyour string is :%s\n",string);
}
```

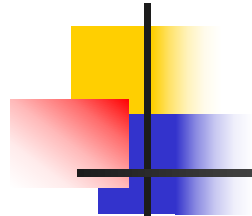


(1)使用调试参数 **-g** 编译file.c源程序，编译之后运行，在提示符中输入字符串“hello world!” 后回车，如图所示

```
lex@lex-virtual-machine:~/lesson/lesson7$ gcc file.c -g -o file
file.c: In function 'main':
file.c:7:2: warning: implicit declaration of function 'gets'; did you mean 'fgets'? [-Wimplicit-function-declaration]
   7 |     gets(string);
     |     ^~~~~
     |     fgets
/usr/bin/ld: /tmp/ccvjrQ1g.o: in function 'main':
/home/lex/lesson/lesson7/file.c:7: warning: the 'gets' function is dangerous and should not be used.
```

```
ld not be used.
lex@lex-virtual-machine:~/lesson/lesson7$ ./file
please input a stringhello world
Segmentation fault (core dumped)
```

原因：由于程序使用了一个未经过初始化的字符型指针string，在执行过程中出现Segmentation fault（段）错误。



(2)查找该程序中出现的问題，利用gdb调试该程序，运行gdb file命令，装入file可执行文件，如图所示：

```
example example.c file.c hello.c hello.out
lex@lex-virtual-machine:~/lesson/lesson7$ gdb file
GNU gdb (Ubuntu 9.2-0ubuntu2) 9.2
Copyright (C) 2020 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GNU Webpage and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from file...
(gdb) █
```


(3)run执行装入的file文件，并使用where 命令查看程序出错的位置，如下图所示：

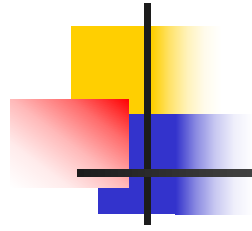
```
Reading symbols from file...
(gdb) run
Starting program: /home/lex/lesson/lesson7/file
please input a string:hello world

Program received signal SIGSEGV, Segmentation fault.
_IO_gets (buf=0x0) at iogets.c:53
53      iogets.c: No such file or directory.
(gdb) where
#0  _IO_gets (buf=0x0) at iogets.c:53
#1  0x0000555555555196 in main () at file.c:7
(gdb)
```

```
1 #include <stdio.h>
2 static char buff[256];
3 static char* string;
4 int main()
5 {
6     printf("please input a string");
7     gets(string);
8     printf("\n your string is :%s \n",string);
9 }
```

(4)或者利用list命令查看调用gets() 函数附近的代码，如图所示：

```
(gdb) list file.c:7
warning: Source file is more recent than executable.
2      static char buff[256];
3      static char* string;
4      int main()
5      {
6          printf("please input a string");
7          gets(string);
8          printf("\n your string is :%s \n",string);
9      }
```

(5)导致gets函数出错的因素就是变量 string，用print命令查看string的值。

```
9      }
(gdb) print string
$1 = 0x0
```

(6)显然string的值是不正确的，指针string应该指向字符数组buff[]的首地址。

◆ 在gdb中，可以直接修改变量的值，在第7行处设置断点break 7，程序重新运行到第7行处停止，可以用set variable命令修改string 的取值,使用next单步执行，将会得到正确的程序运行结果

```
(gdb) break 7
Breakpoint 1 at 0x55555555182: file file.c, line 7.
(gdb) run
The program being debugged has been started already.
Start it from the beginning? (y or n) y
Starting program: /home/lex/lesson/lesson7/file

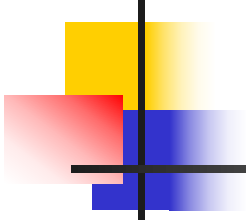
Breakpoint 1, main () at file.c:7
7      gets(string);
(gdb) set variable string=buff
(gdb) next
please input a string:hello world
8      printf("\n your string is :%s \n",string);
(gdb) next

your string is ::hello world
9      }
```



说明：

- 如果步骤（4）利用list命令查看调用gets() 函数附近的代码，没有显示出来代码的内容，list 后面加上文件名和行号，如下图所示。
- 一般的，找到错误后从*.c文件直接更改错误后再运行程序即可。



```

lex@lex-virtual-machine:~/lesson/lesson7$ gcc -g example3.c -o example3
example3.c:13:1: warning: return type defaults to 'int' [-Wimplicit-int]
   13 | main()
       | ~~~~~
example3.c: In function 'main':
example3.c:21:27: warning: format '%d' expects argument of type 'int', but argument
has type 'long int' [-Wformat=]
   21 | printf("result[1-100] = %d /n",result);
       |                ~^          ~~~~~
               |             |
               int          long int
               %ld
lex@lex-virtual-machine:~/lesson/lesson7$ gdb example3
GNU gdb (Ubuntu 8.2-2ubuntu2) 8.2

```



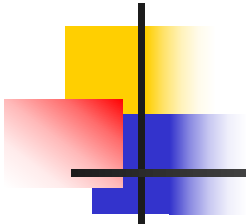
loading symbols from example3.elf

```
(gdb) l
1      #include <stdio.h>
2
3      int func(int n)
4      {
5          int sum=0,i;
6          for(i=0;i<n;i++)
7          {
8              sum+=i;
9          }
10         return sum;
(gdb) l
11     }
12
13     main()
14     {
15         int i;
16         long result=0;
17         for(i=1;i<=100;i++)
18         {
19             result+=i;
20         }
(gdb) l
21         printf("result[1-100] = %d /n",result);
22         printf("result[1-200] = %d /n",func(250));
23     }
```

```
(gdb) break 15
Breakpoint 1 at 0x1187: file example3.c, line 16.
(gdb) break func
Breakpoint 2 at 0x1149: file example3.c, line 4.
(gdb) info break
Num      Type           Disp Enb Address            What
1        breakpoint     keep y   0x00000000000001187 in
2        breakpoint     keep y   0x00000000000001149 in
(gdb) r
Starting program: /home/lex/lesson/lesson7/example3

Breakpoint 1, main () at example3.c:16
16          long result=0;
(gdb) n
17          for(i=1;i<=100;i++)
(gdb) n
19              result+=i;
(gdb) n
17          for(i=1;i<=100;i++)
(gdb) n
19              result+=i;
(gdb) c
Continuing.

Breakpoint 2, func (n=21845) at example3.c:4
```



```

(gdb) n
5           int sum=0,i;
(gdb) n
6           for(i=0;i<n;i++)
(gdb) p i
$1 = 0
(gdb) n
8           sum+=i;
(gdb) n
6           for(i=0;i<n;i++)
(gdb) psum
Undefined command: "psum". Try "help".
(gdb) p sum
$2 = 0
(gdb) n
8           sum+=i;
(gdb) n
6           for(i=0;i<n;i++)
(gdb) p sum
$3 = 1
(gdb) p sum
$4 = 1
(gdb) n
8           sum+=i;
(gdb) p sum
$5 = 1

```

```

(gdb) n
6           for(i=0;i<n;i++)
(gdb) n
8           sum+=i;
(gdb) n
6           for(i=0;i<n;i++)
(gdb) p sum
$6 = 6
(gdb) bt
#0 func (n=250) at example3.c:6
#1 0x000055555555551cd in main () at example3.c:22
(gdb) finish
Run till exit from #0 func (n=250) at example3.c:6
0x000055555555551cd in main () at example3.c:22
22 printf("result[1-200] = %d /n",func(250));
Value returned is $7 = 31125
(gdb) c
Continuing.
result[1-100] = 5050 /nresult[1-200] = 31125 /n[Inferior 1
ally]
(gdb) r
Starting program: /home/lex/lesson/lesson7/example3

Breakpoint 1, main () at example3.c:16

```



```
Starting program: /home/lex/lesson/lesson7/example3
```

```
Breakpoint 1, main () at example3.c:16
```

```
16          long result=0;
```

```
(gdb) watch i
```

```
Hardware watchpoint 3: i
```

```
(gdb) c
```

```
Continuing.
```

```
Hardware watchpoint 3: i
```

```
Old value = 32767
```

```
New value = 1
```

```
main () at example3.c:17
```

```
17          for(i=1;i<=100;i++)
```

```
(gdb) watch i
```

```
Hardware watchpoint 4: i
```

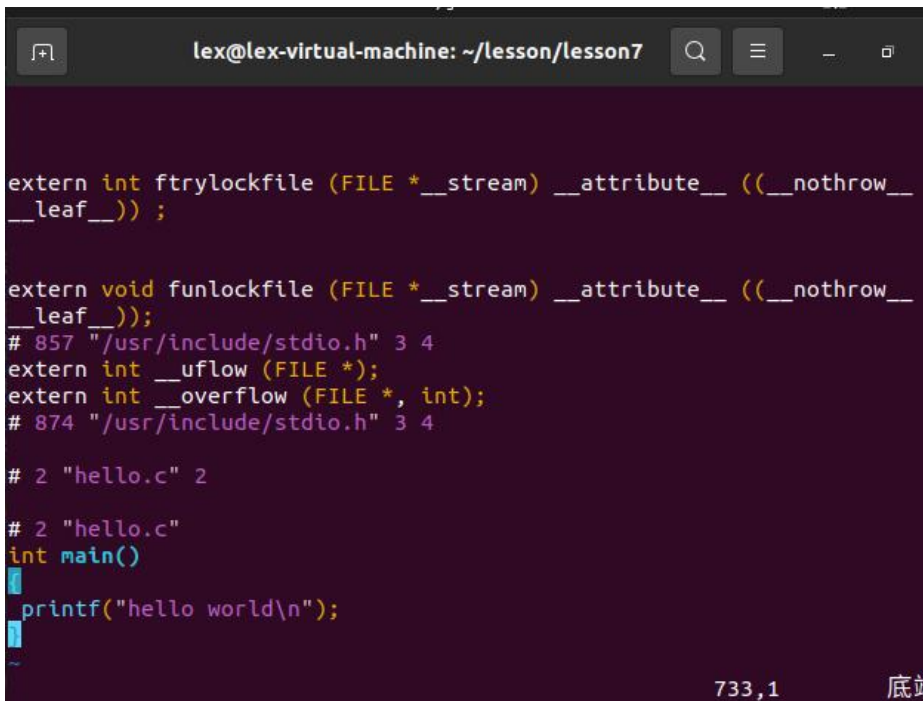
```
(gdb) █
```




1.预处理/预编译hello.c程序，将预编译结果输出到hello.i。

■ 执行的命令：`gcc -E hello.c -o hello.i` #-E'关键字对应着“预处理后即停止，不进行编译、汇编及链接”

■ 在预编译的过程中，gcc对源文件所包含的头文件stdio.h进行了预处理。图为hello.i部分内容



```
lex@lex-virtual-machine: ~/lesson/lesson7

extern int ftrylockfile (FILE *__stream) __attribute__ ((__nothrow__
__leaf__));

extern void funlockfile (FILE *__stream) __attribute__ ((__nothrow__
__leaf__));
# 857 "/usr/include/stdio.h" 3 4
extern int __uflow (FILE *);
extern int __overflow (FILE *, int);
# 874 "/usr/include/stdio.h" 3 4

# 2 "hello.c" 2

# 2 "hello.c"
int main()
{
    printf("hello world\n");
}
```

[返回](#)

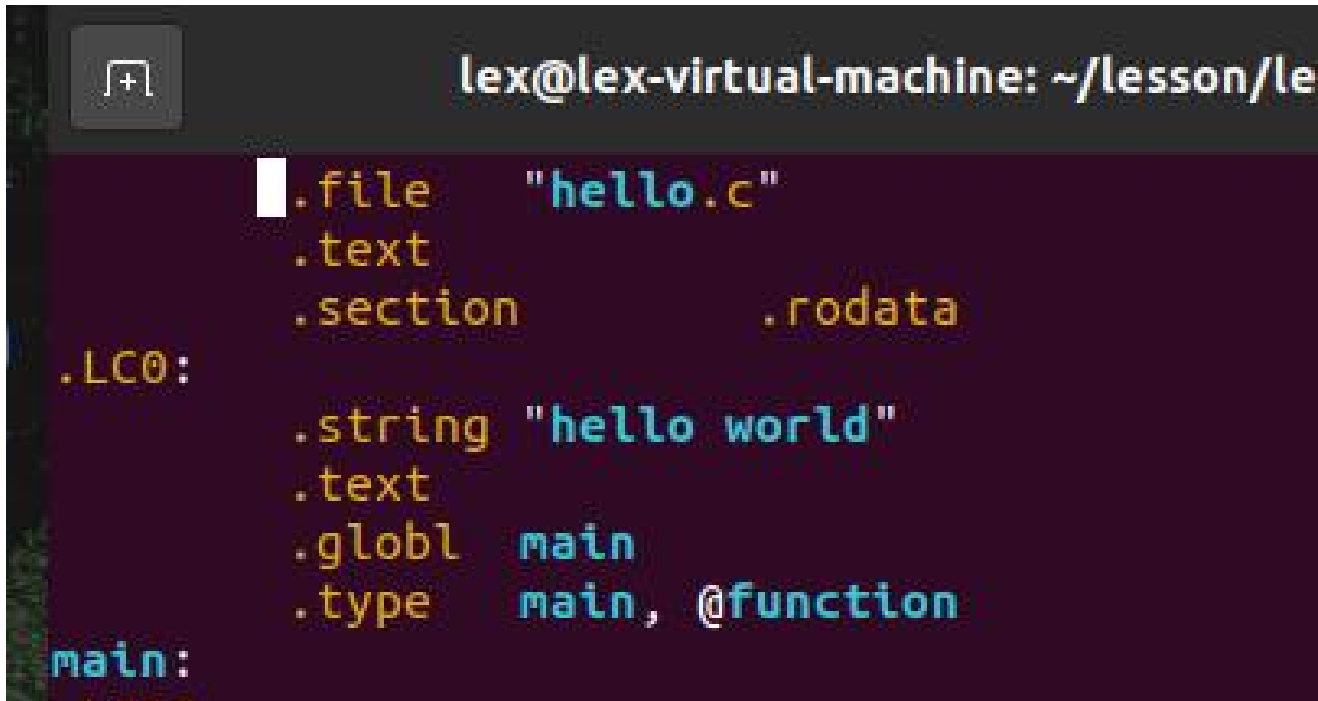


2.编译

例：编译hello.i文件，编译后生成汇编语言文件hello.s。

执行的gcc命令：`gcc -S hello.i -o hello.s`

hello.s就是生成的汇编语言文件，如图所示。



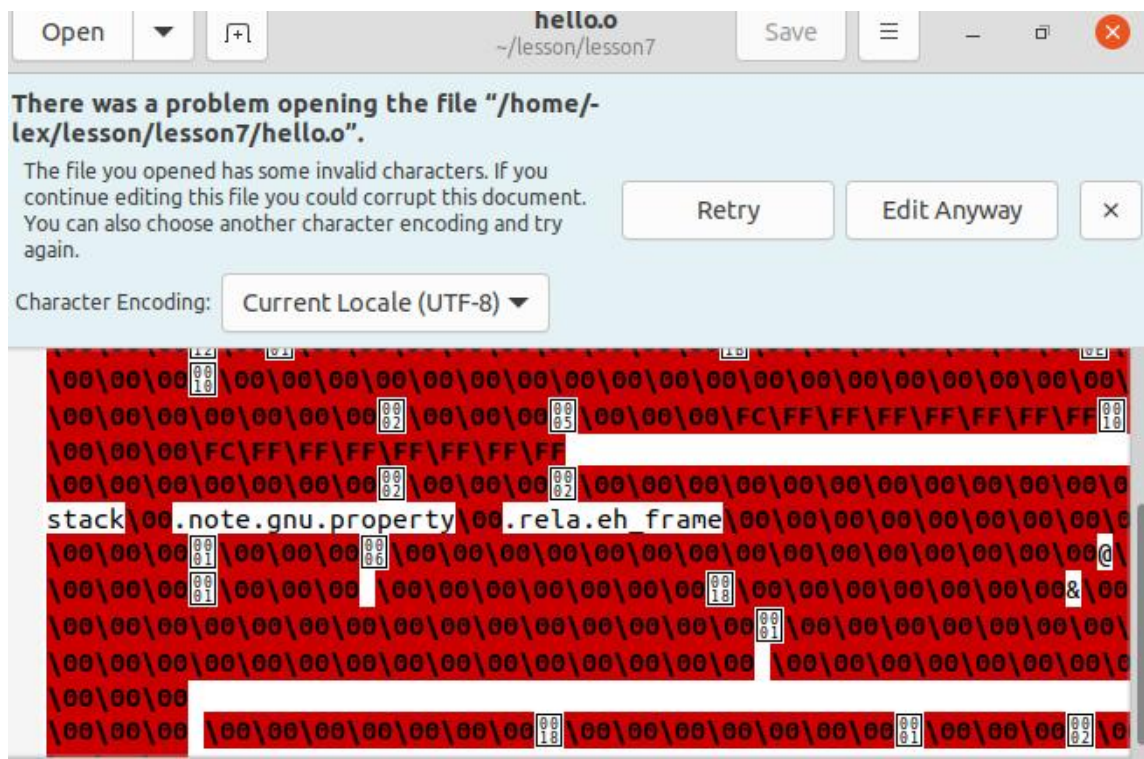
```
lex@lex-virtual-machine: ~/lesson/le
.file    "hello.c"
.text
.section     .rodata
.LC0:
.string "hello world"
.text
.globl  main
.type   main, @function
main:
```

[返回](#)

3.汇编：将输入的汇编语言文件转换为目标代码，可以使用-c选项完成。

例：将汇编语言文件hello.s转换为目标程序hello.o,在Linux系统上生成ELF目标文件 (OBJ文件Executable and Linkable Format)。

执行gcc命令：`gcc -c hello.s -o hello.o`



[返回](#)



4.链接

例：将目标程序hello.o连接成可执行文件hello.out。

执行gcc命令为：gcc hello.o -o

```
lex@lex-virtual-machine:~/lesson/lesson7$ gcc hello.o
lex@lex-virtual-machine:~/lesson/lesson7$ gcc hello.o -o hello.out
lex@lex-virtual-machine:~/lesson/lesson7$ ls -l
total 52
-rw-rw-r-- 1 lex lex    60  4月 22 19:41 hello.c
-rw-rw-r-- 1 lex lex 16641  4月 22 20:02 hello.i
-rw-rw-r-- 1 lex lex  1680  4月 22 20:23 hello.o
-rwxrwxr-x 1 lex lex 16808  4月 22 20:34 hello.out
-rw-rw-r-- 1 lex lex   649  4月 22 20:19 hello.s
lex@lex-virtual-machine:~/lesson/lesson7$ ./hello.out
hello world
lex@lex-virtual-machine:~/lesson/lesson7$
```

[返回](#)