

homework1

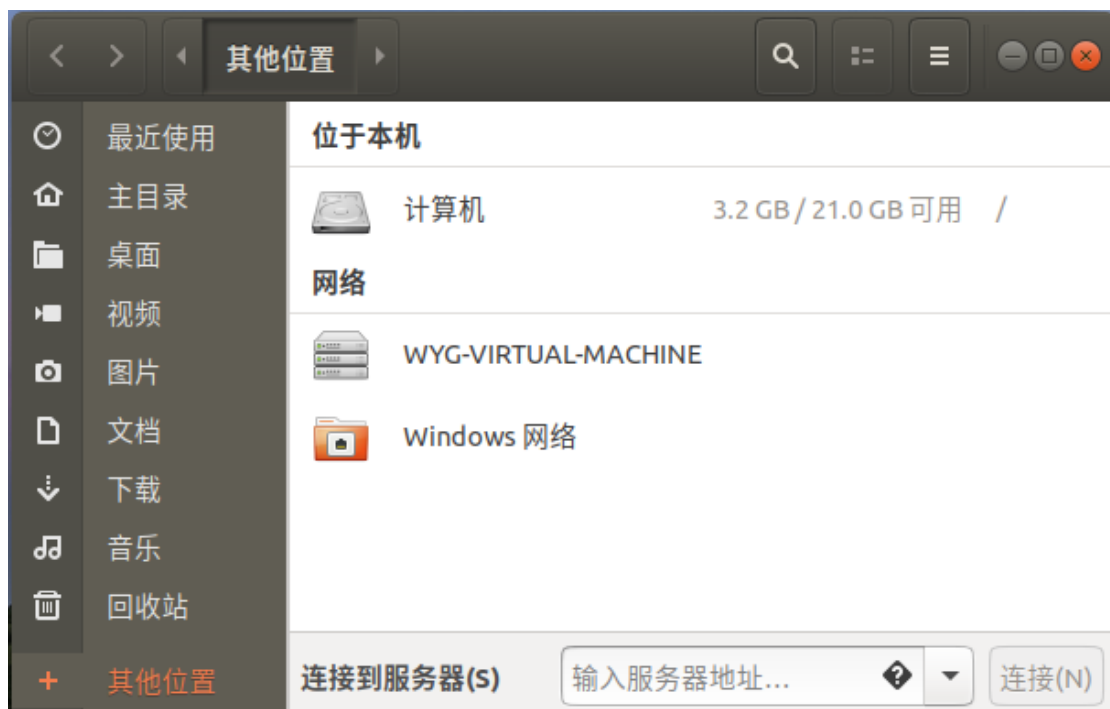
❖ 安装 Ubuntu14.04 或以上版本的 linux 系统并熟悉基本操作;

(一) 查看系统目录结构

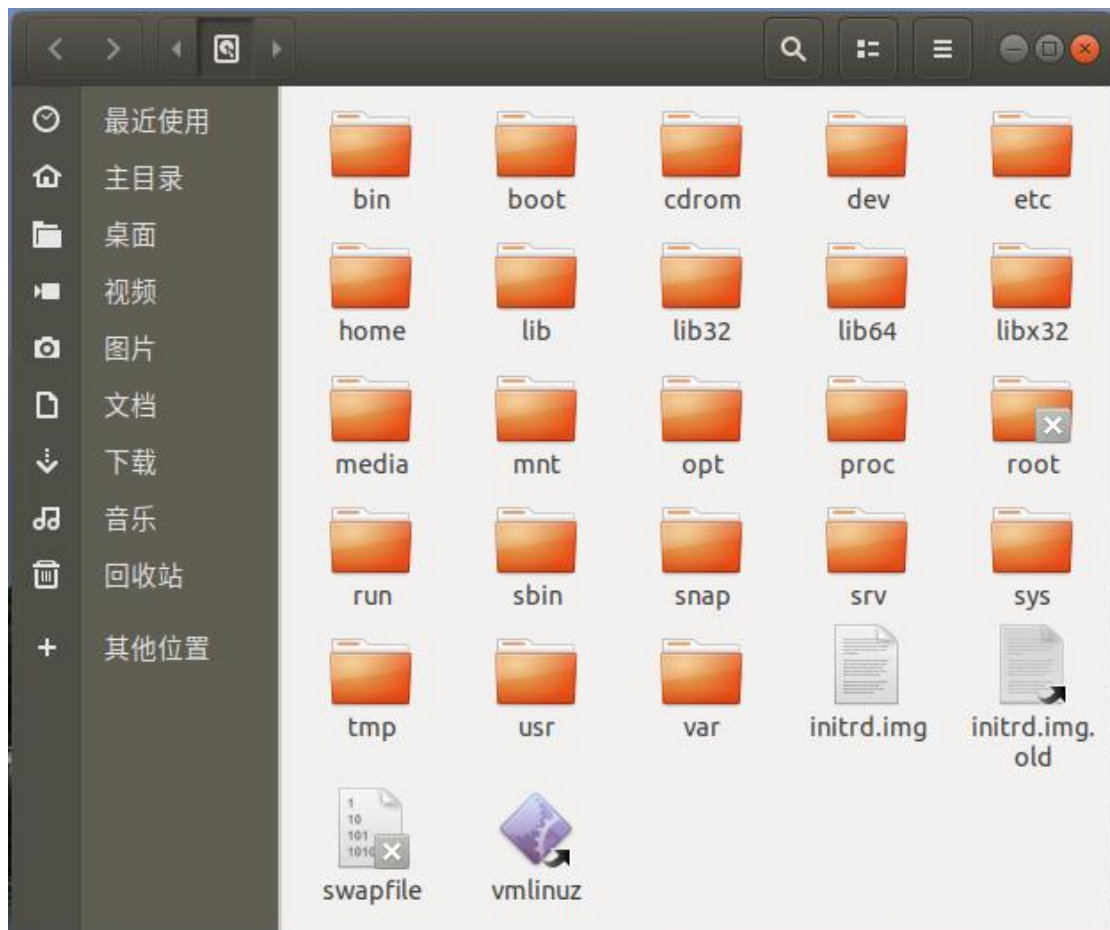
1) 查看桌面在系统目录的位置，并观察目录结构

```
wyg@wyg-virtual-machine: /
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)
(base) wyg@wyg-virtual-machine:~/桌面$ pwd
/home/wyg/桌面
(base) wyg@wyg-virtual-machine:~/桌面$ ls -a
.  ..  bash.doc  Robot  tool
(base) wyg@wyg-virtual-machine:~/桌面$ cd ..
(base) wyg@wyg-virtual-machine:~$ ls
anaconda3  snap  公共的  视频  文档  音乐
examples.desktop  sys  模板  图片  下载  桌面
(base) wyg@wyg-virtual-machine:~$ cd ..
(base) wyg@wyg-virtual-machine:/home$ ls
wyg
(base) wyg@wyg-virtual-machine:/home$ cd ..
(base) wyg@wyg-virtual-machine:/$ ls
bin      etc          lib          lost+found  proc      snap         tmp
boot     home         lib32        media       root      srv          usr
cdrom    initrd.img  lib64        mnt         run       swapfile    var
dev      initrd.img.old  libx32      opt         sbin      sys         vmlinuz
(base) wyg@wyg-virtual-machine:/$ cd ..
(base) wyg@wyg-virtual-machine:/$ ls
bin      etc          lib          lost+found  proc      snap         tmp
boot     home         lib32        media       root      srv          usr
cdrom    initrd.img  lib64        mnt         run       swapfile    var
dev      initrd.img.old  libx32      opt         sbin      sys         vmlinuz
(base) wyg@wyg-virtual-machine:/$
```

2) 查看所有位置的存储设备



3) Linux 的根目录



(1) /bin 用户二进制文件：包含二进制可执行文件，系统所有用户可执行文件都在这个文件夹里，例如：ls, cp, ping 等。

(2) /sbin 系统二进制文件：包含二进制可执行文件，但只能由系统管理员运行，对系统进行维护。

(3) /etc 配置文件：包含所有程序配置文件，也包含了用于启动/停止单个程序的启动和关闭 shell 脚本。

(4) /dev 设备文件：包含终端所有设备，USB 或连接到系统的任何设备。

(5) /proc 进程信息：包含系统进程的相关信息。

(6) /var 变量文件：可以找到内容可能增长的文件。

(7) /tmp 临时文件：包含系统和用户创建的临时文件。系统重启时，目录清空。

(8) /usr 用户程序

包含二进制文件、库文件、文档和二级程序的源代码。

/usr/bin 中包含用户程序的二进制文件。如果你在/bin 中找不到用户二进制文件，到/usr/bin 目录看看。例如：at、awk、cc、less、scp。

/usr/sbin 中包含系统管理员的二进制文件。如果你在/sbin 中找不到系统二进制文件，到/usr/sbin 目录看看。例如：atd、cron、sshd、useradd、userdel。

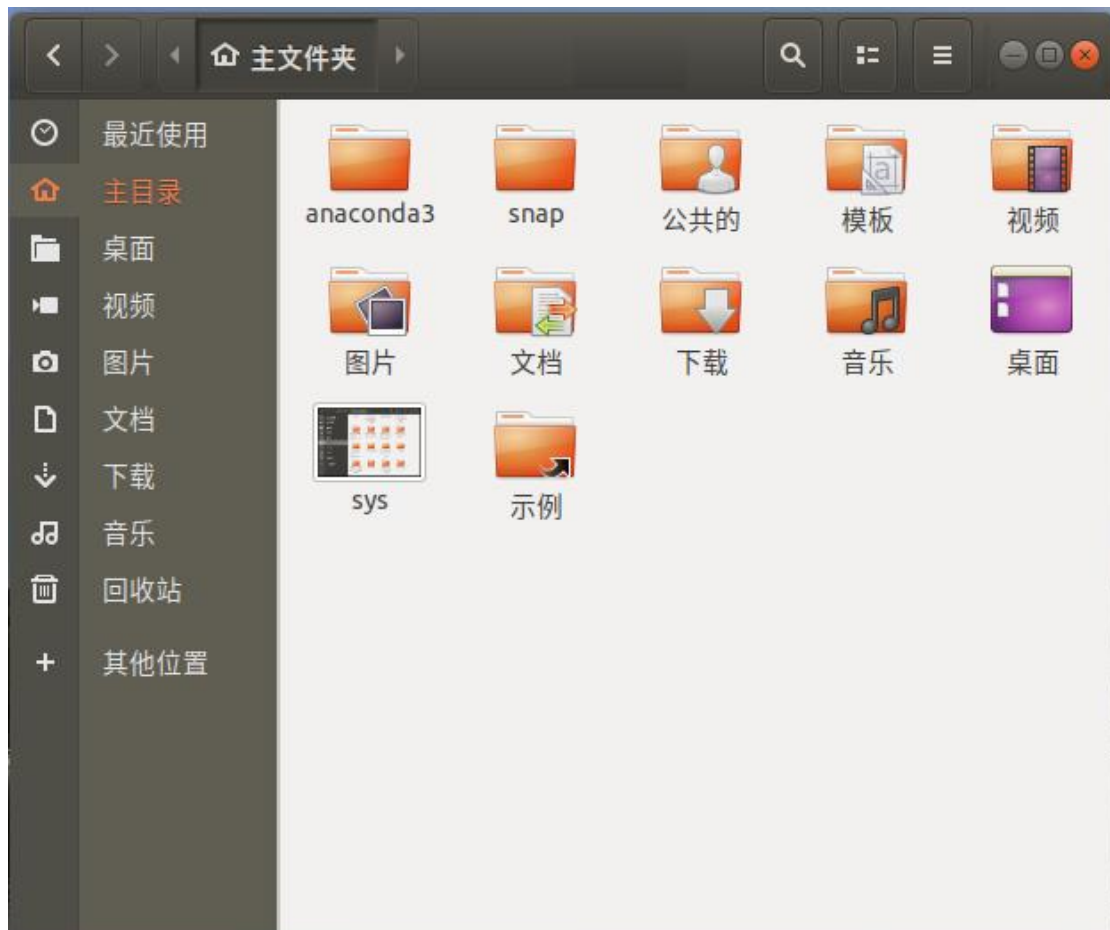
/usr/lib 中包含了/usr/bin 和/usr/sbin 用到的库。

/usr/local 中包含了从源安装的用户程序。例如，当你从源安装 Apache，它会在/usr/local/apache2 中。

(9) /home HOME 目录：所有用户用来存档他们的个人档案。

- (10) /boot 引导加载程序文件：包含引导加载程序相关的文件。
内核的 initrd、vmlinuz、grub 文件位于/boot 下。
- (11) /lib 系统库：包含支持位于/bin 和/sbin 下的二进制文件的库文件。
库文件名为 ld 或 lib.so.*
- (12) /opt 可选的附加应用程序：opt 代表 optional；
包含从个别厂商的附加应用程序。
附加应用程序应该安装在/opt/或者/opt/的子目录下。
- (13) /mnt 挂载目录：临时安装目录，系统管理员可以挂载文件系统。
- (14) /media 可移动媒体设备：用于挂载可移动设备的临时目录。
举例来说，挂载 CD-ROM 的/media/cdrom，挂载软盘驱动器的/media/floppy；
- (15) /srv 服务数据：srv 代表服务。
包含服务器特定服务相关的数据。
例如，/srv/cvs 包含 cvs 相关的数据。

4) 主文件夹，位于根目录的/home/username 下：所有用户用来存档他们的个人档案，每个用户对应一个主文件夹。



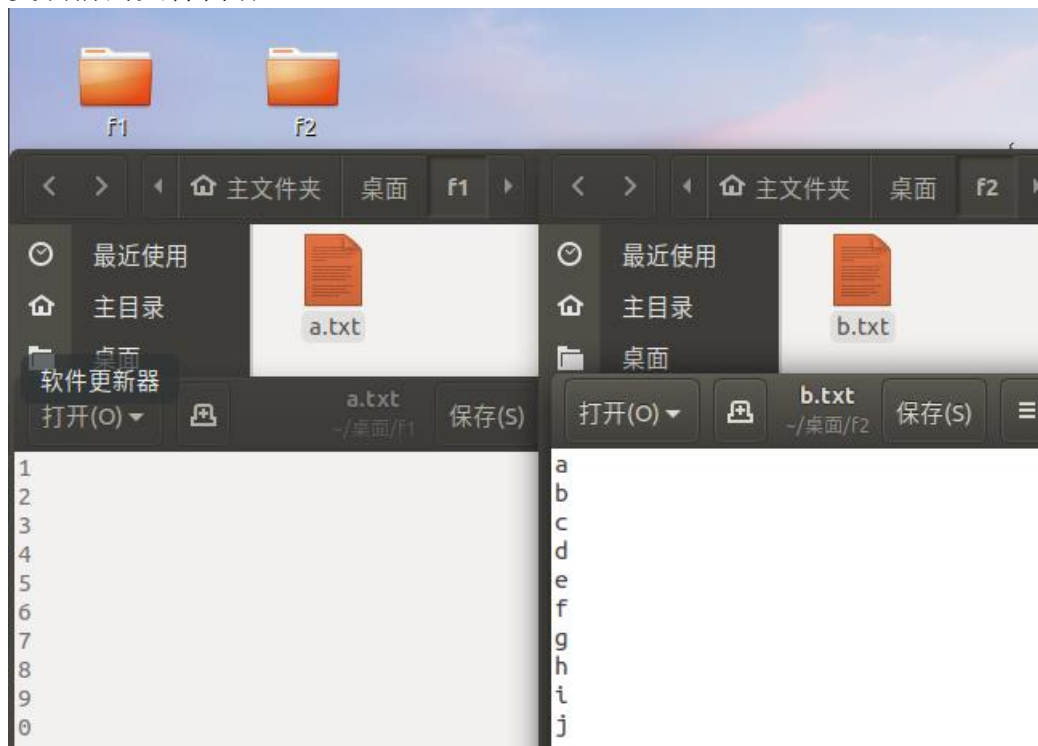
都是按照功能来命名，文件夹名称语言随系统语言同步变化

(二) Linux 基本操作

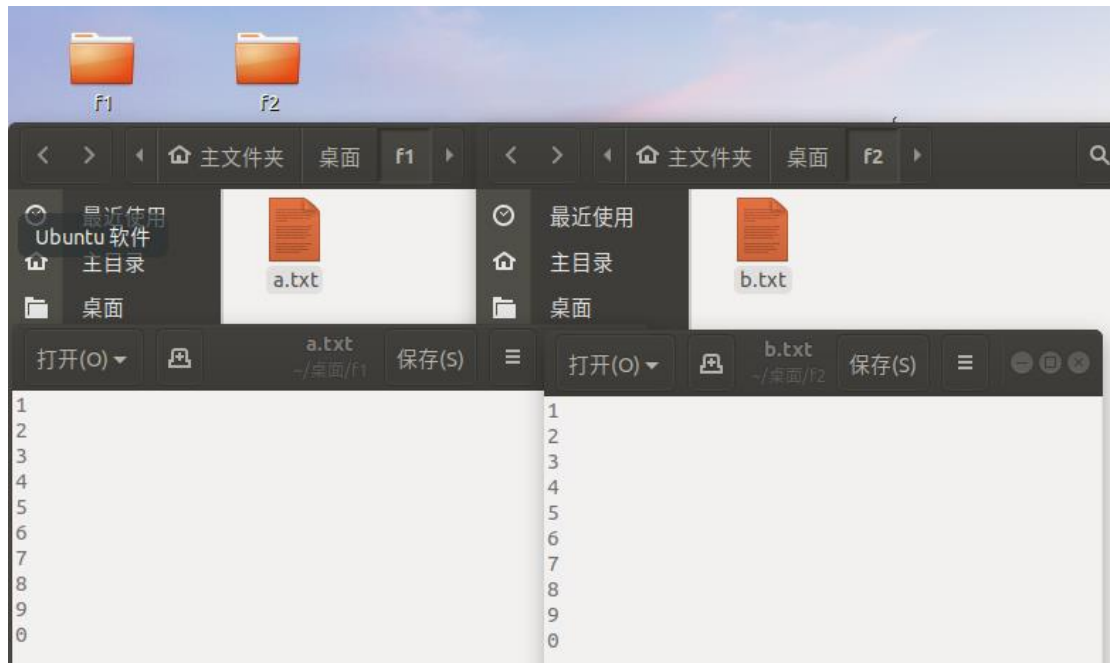
1) 增删查改文件或文件夹

```
wyg@wyg-virtual-machine: ~/桌面
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)
(base) wyg@wyg-virtual-machine:~$ mkdir f1
(base) wyg@wyg-virtual-machine:~$ pwd
/home/wyg
(base) wyg@wyg-virtual-machine:~$ rm -rf f1
(base) wyg@wyg-virtual-machine:~$ ls
anaconda3      snap  公共的  视频  文档  音乐
examples.desktop  sys  模板  图片  下载  桌面
(base) wyg@wyg-virtual-machine:~$ cd 桌面
(base) wyg@wyg-virtual-machine:~/桌面$ mkdir f1
(base) wyg@wyg-virtual-machine:~/桌面$ mkdir f2
(base) wyg@wyg-virtual-machine:~/桌面$ cd f1
(base) wyg@wyg-virtual-machine:~/桌面/f1$ touch a.txt
(base) wyg@wyg-virtual-machine:~/桌面/f1$ cd ..
(base) wyg@wyg-virtual-machine:~/桌面$ cd f2
(base) wyg@wyg-virtual-machine:~/桌面/f2$ touch b.txt
(base) wyg@wyg-virtual-machine:~/桌面/f2$ cd ..
(base) wyg@wyg-virtual-machine:~/桌面$ cp /f1/a.txt /f2/b.txt
cp: 无法获取 '/f1/a.txt' 的文件状态(stat): 没有那个文件或目录
(base) wyg@wyg-virtual-machine:~/桌面$ cp ./f1/a.txt ./f2/b.txt
(base) wyg@wyg-virtual-machine:~/桌面$ head ./f1/a.txt
1
2
3
4
```

2) 练习：将 f1 文件夹中的 a.txt 的内容复制到 f2 文件夹中的 b.txt 中 复制前的文件内容



复制后的文件内容



查看文件的前十行

```
wyg@wyg-virtual-machine: ~/桌面
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)
(base) wyg@wyg-virtual-machine:~/桌面$ head ./f2/b.txt
a
b
c
d
e
f
g
h
i
j
(base) wyg@wyg-virtual-machine:~/桌面$ cp ./f1/a.txt ./f2/b.txt
(base) wyg@wyg-virtual-machine:~/桌面$ head ./f2/b.txt
1
2
3
4
5
6
7
8
9
0
(base) wyg@wyg-virtual-machine:~/桌面$ rm -rf f1
(base) wyg@wyg-virtual-machine:~/桌面$ rm -rf f2
```

3) 查看日期与时间

```
/home/wyg
(base) wyg@wyg-virtual-machine:~$ date
2019年 11月 23日 星期六 19:32:56 CST
(base) wyg@wyg-virtual-machine:~$ cal
      十一月 2019
日 一 二 三 四 五 六
                1  2
 3  4  5  6  7  8  9
10 11 12 13 14 15 16
17 18 19 20 21 22 23
24 25 26 27 28 29 30
```


(三) Vim 编辑器

1) 删除 vi 编辑器，安装 vim 编辑器

```
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)
(base) wyg@wyg-virtual-machine:~$ sudo apt-get remove vim-common
[sudo] wyg 的密码:
正在读取软件包列表... 完成
正在分析软件包的依赖关系树
正在读取状态信息... 完成
下列软件包是自动安装的并且现在不需要了:
  vim-runtime
使用'sudo apt autoremove'来卸载它(它们)。
下列软件包将被【卸载】:
  vim vim-common
升级了 0 个软件包，新安装了 0 个软件包，要卸载 2 个软件包，有 334 个软件包未被升级。
解压缩后将会空出 3,189 kB 的空间。
您希望继续执行吗? [Y/n] y
(正在读取数据库 ... 系统当前共安装有 161005 个文件和目录。)
正在卸载 vim (2:8.0.1453-1ubuntu1.1) ...
正在卸载 vim-common (2:8.0.1453-1ubuntu1.1) ...
正在处理用于 mime-support (3.60ubuntu1) 的触发器 ...
正在处理用于 desktop-file-utils (0.23-1ubuntu3.18.04.1) 的触发器 ...
正在处理用于 man-db (2.8.3-2) 的触发器 ...
正在处理用于 gnome-menus (3.13.3-11ubuntu1) 的触发器 ...
正在处理用于 hicolor-icon-theme (0.17-2) 的触发器 ...
(base) wyg@wyg-virtual-machine:~$ sudo apt-get install vim
正在读取软件包列表... 完成
正在分析软件包的依赖关系树
正在读取状态信息... 完成
将会同时安装下列软件:
  vim-common
建议安装:
  ctags vim-doc vim-scripts
下列【新】软件包将被安装:
  vim vim-common
升级了 0 个软件包，新安装了 2 个软件包，要卸载 0 个软件包，有 334 个软件包未被升级。
需要下载 0 B/1,222 kB 的归档。
解压缩后会消耗 3,189 kB 的额外空间。
您希望继续执行吗? [Y/n] y
```

2) 进入 vim 编辑器教程

```
(base) wyg@wyg-virtual-machine:~$ vimtutor
(base) wyg@wyg-virtual-machine:~$
```

进去了解了了解了集本操作，就是移动光标有点不习惯，感觉鼠标点击会快一些

version 1.1
April 1st, 06
翻译: 2006-5-21

vi / vim 键盘图

Esc
命令
模式

~ 切换大小写
! 外部过滤器
@ 运行宏
prev ident
\$ 行尾
% 括号匹配
^ "找" 行首
& 重复
* next ident
(句首
) 下一句首
"soft" bol
+ 后一行
- 前一行
= 自动格式化

Q 切换到 ex 模式
q 复制宏
W 单词
w 单词
E 词尾
e 词尾
R 替换模式
r 替换字符
T back till
t till
Y 拷贝行
y 拷贝
U 取消行内命令
u 取消命令
I 到行首插入
i 插入模式
O 分段(前)
o 分段(后)
P 粘贴
p 粘贴
{ 段首
} 段尾
[杂项
] 杂项

A 在行尾附加
a 附加
S 删除行并插入
s 删除字符并插入
D 删除全行
d 删除
F 行内字符向前查找
f 行内字符向后查找
G 文尾/行号
g 附加命令
H 屏幕前行
h 左
J 合并两行
j 下
K 帮助
k 上
L 屏幕后行
l 右
: ex 命令
; 重复
/ 向前搜索
" 寄存器
' 寄存器
~ 行首/行尾
! 未用!

Z 退出
Z 附加命令
X 退格
x 删除(字符)
C 修改整行
c 修改
V 可视模式
v 可视模式
B 前一单词
b 前一单词
N 查找下一处
n 查找下一处
M 屏幕中行
m 设置标志
< 反缩进
> 缩进
? 向前搜索
/ 向后搜索

动作 移动光标，或者定义操作的范围
命令 直接执行的命令，红色命令进入编辑模式
操作 后面跟数字表示操作范围的指令
extra 特殊功能，需要额外的输入
q 后缀数字参数

w,e,b 命令
小写(b): quux(foo, bar, baz)
大写(B): quux(Foo, Bar, Baz)

主要 ex 命令:
:w (保存), :q (退出), :q! (不保存退出)
:e f (打开文件 f)
:%s/s/y/g ('y' 全局替换 's')
:h (帮助 in vim), :new (新建文件 in vim)
其它重要命令:
CTRL-R: 重复 (vim),
CTRL-F/-B: 上翻/下翻,
CTRL-E/-Y: 上滚/下滚,
CTRL-V: 块可视模式 (vim only)
可视模式:
漫游后对选中的区域执行操作 (vim only)

备注:
(1) 在 拷贝/粘贴/删除 命令前使用 "x (x=a..z)" 使用命令的寄存器(剪贴板)
(如: "ay8 拷贝剩余的行内容至寄存器 'a')
(2) 命令前添加数字 多遍重复操作 (e.g.: 2p, d2w, 5i, d4j)
(3) 重复本字符在光标所在行执行操作 (dd = 删除本行, >> = 行首缩进)
(4) ZZ 保存退出, ZQ 不保存退出
(5) rt: 移动光标所在行至屏幕顶端, zb: 底端, zz: 中间
(6) gg: 文首 (vim only), gG: 打开光标处的文件名 (vim only)

原图: www.viemu.com 翻译: fdl (linuxsir)

(四) 安装 gcc&g++

```
(base) wyg@wyg-virtual-machine:~$ sudo apt-get install build-essential
正在读取软件包列表... 完成
正在分析软件包的依赖关系树
正在读取状态信息... 完成
build-essential 已经是最新版 (12.4ubuntu1)。
升级了 0 个软件包，新安装了 0 个软件包，要卸载 0 个软件包，有 334 个软件包未被升级。
(base) wyg@wyg-virtual-machine:~$ make -v
GNU Make 4.1
为 x86_64-linux-gnu 编译
Copyright (C) 1988-2014 Free Software Foundation, Inc.
许可证: GPLv3+: GNU 通用公共许可证第 3 版或更新版本<http://gnu.org/licenses/gpl.html>。
本软件是自由软件：您可以自由修改和重新发布它。
在法律允许的范围内没有其他保证。
(base) wyg@wyg-virtual-machine:~$ gcc -v
Using built-in specs.
COLLECT_GCC=gcc
COLLECT_LTO_WRAPPER=/usr/lib/gcc/x86_64-linux-gnu/7/lto-wrapper
OFFLOAD_TARGET_NAMES=nvptx-none
OFFLOAD_TARGET_DEFAULT=1
Target: x86_64-linux-gnu
Configured with: ../src/configure -v --with-pkgversion='Ubuntu 7.4.0-1ubuntu1~18.04.1' --
l=file:///usr/share/doc/gcc-7/README.Bugs --enable-languages=c,ada,c++,go,brig,d,fortran,
++ --prefix=/usr --with-gcc-major-version-only --program-suffix=-7 --program-prefix=x86_6
u- --enable-shared --enable-linker-build-id --libexecdir=/usr/lib --without-included-gett
le-threads=posix --libdir=/usr/lib --enable-nls --with-sysroot=/ --enable-clocale=gnu --e
tdcxx-debug --enable-libstdcxx-time=yes --with-default-libstdcxx-abi=new --enable-gnu-uni
--disable-vtable-verify --enable-libmpx --enable-plugin --enable-default-pie --with-syst
with-target-system-zlib --enable-objc-gc=auto --enable-multiarch --disable-werror --with-
86 --with-abi=m64 --with-multilib-list=m32,m64,mx32 --enable-multilib --with-tune=generic
offload-targets=nvptx-none --without-cuda-driver --enable-checking=release --build=x86_64
--host=x86_64-linux-gnu --target=x86_64-linux-gnu
Thread model: posix
gcc version 7.4.0 (Ubuntu 7.4.0-1ubuntu1~18.04.1)
(base) wyg@wyg-virtual-machine:~$
```

(五) 安装 Eigen

```
(base) wyg@wyg-virtual-machine:~$ sudo apt-get install libeigen3-dev
正在读取软件包列表... 完成
正在分析软件包的依赖关系树
正在读取状态信息... 完成
libeigen3-dev 已经是最新版 (3.3.4-4)。
升级了 0 个软件包，新安装了 0 个软件包，要卸载 0 个软件包，有 334 个软件包未被升级。
(base) wyg@wyg-virtual-machine:~$ sudo updatedb
(base) wyg@wyg-virtual-machine:~$ locate eigen3
/home/wyg/anaconda3/lib/python3.7/site-packages/tensorflow/include/third_party/eigen3
/home/wyg/anaconda3/lib/python3.7/site-packages/tensorflow/include/third_party/eigen3/Eigen
/home/wyg/anaconda3/lib/python3.7/site-packages/tensorflow/include/third_party/eigen3/LICENSE
/home/wyg/anaconda3/lib/python3.7/site-packages/tensorflow/include/third_party/eigen3/unsupported
/home/wyg/anaconda3/lib/python3.7/site-packages/tensorflow/include/third_party/eigen3/Eigen/Cholesk
```

❖ 书写一个由 cmake 组织的 C++ 工程，并书写

CMakeLists.txt，要求如下：

1. 参考示例代码，编译 include/hello.h 和 src/hello.c 构成 libhello.so 库。

其中 hello.c 中提供一个函数 sayHello()，调用此函数时往屏幕输出一行“Hello”

2. 文件 useHello.c 中含有一个 main 函数，它可以编译成一个可执行文件，名为“sayhello”。

3. 默认用 Release 模式编译这个工程。

4. 支持使用命令 sudo make install，该命令将 hello.h 放

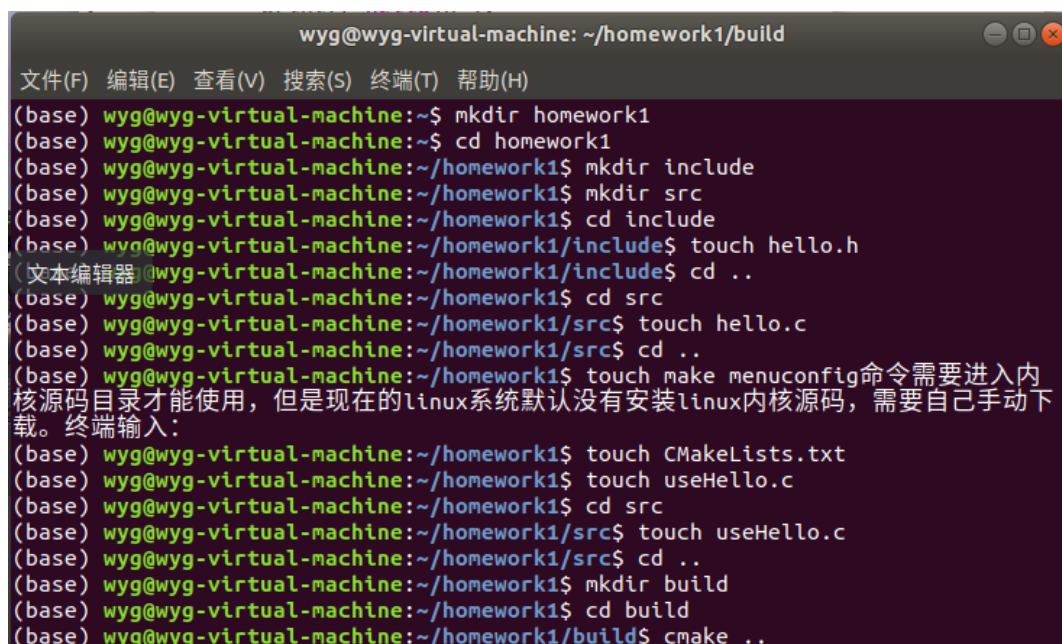
至/usr/local/include/下，将 libhello.so 放 至/usr/local/lib/下

1) 准备程序文件

由题意文件目录结构如下：

```
.
├── build
├── CMakeLists.txt
├── include
│   └── hello.h
└── src
    ├── hello.c
    └── useHello.c
```

生成目录结构



```
wyg@wyg-virtual-machine: ~/homework1/build
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)
(base) wyg@wyg-virtual-machine:~$ mkdir homework1
(base) wyg@wyg-virtual-machine:~$ cd homework1
(base) wyg@wyg-virtual-machine:~/homework1$ mkdir include
(base) wyg@wyg-virtual-machine:~/homework1$ mkdir src
(base) wyg@wyg-virtual-machine:~/homework1$ cd include
(base) wyg@wyg-virtual-machine:~/homework1/include$ touch hello.h
(文本编辑器)wyg@wyg-virtual-machine:~/homework1/include$ cd ..
(base) wyg@wyg-virtual-machine:~/homework1$ cd src
(base) wyg@wyg-virtual-machine:~/homework1/src$ touch hello.c
(base) wyg@wyg-virtual-machine:~/homework1/src$ cd ..
(base) wyg@wyg-virtual-machine:~/homework1$ touch make menuconfig命令需要进入内
核源码目录才能使用，但是现在的linux系统默认没有安装linux内核源码，需要自己手动下
载。终端输入：
(base) wyg@wyg-virtual-machine:~/homework1$ touch CMakeLists.txt
(base) wyg@wyg-virtual-machine:~/homework1$ touch useHello.c
(base) wyg@wyg-virtual-machine:~/homework1$ cd src
(base) wyg@wyg-virtual-machine:~/homework1/src$ touch useHello.c
(base) wyg@wyg-virtual-machine:~/homework1/src$ cd ..
(base) wyg@wyg-virtual-machine:~/homework1$ mkdir build
(base) wyg@wyg-virtual-machine:~/homework1$ cd build
(base) wyg@wyg-virtual-machine:~/homework1/build$ cmake ..
```


头文件 hello.h，如下所示：

```
hello.h
~/homework1/include
保存(S)

#ifndef HELLO_FILE_HEADER_INC
#define HELLO_FIEL_HEADER_INC

void sayHello();

#endif
```

源文件 hello.c，如下所示：

```
hello.c
~/homework1/src
保存(S)

#include "../include/hello.h"
#include <stdio.h>

void sayHello()
{
    printf("hello\n");
}
```

useHello.c 主函数，如下所示：

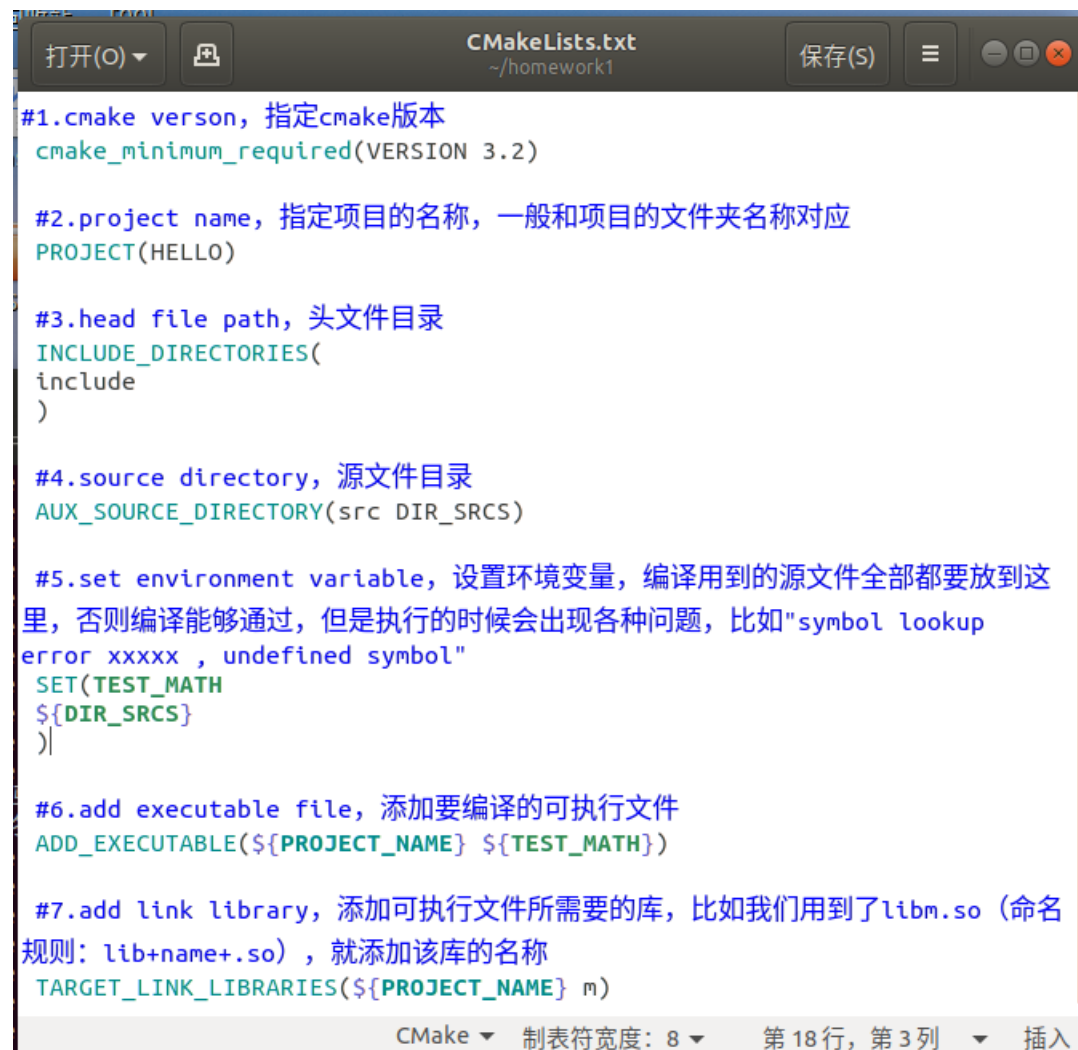
```
useHello.c
~/homework1...
保存(S)

#include "../include/hello.h"
#include <stdio.h>

int main(int argc, char** argv)
{
    sayHello();
    return 0;
}
```

2) 编写 CMakeLists.txt

接下来编写 CMakeLists.txt 文件，该文件放在和 src, include 的同级目录，实际方哪里都可以，只要里面编写的路径能够正确指向就好了。CMakeLists.txt 文件，如下所示：



```
#1.cmake version, 指定cmake版本
cmake_minimum_required(VERSION 3.2)

#2.project name, 指定项目的名称, 一般和项目的文件夹名称对应
PROJECT(HELLO)

#3.head file path, 头文件目录
INCLUDE_DIRECTORIES(
include
)

#4.source directory, 源文件目录
AUX_SOURCE_DIRECTORY(src DIR_SRCS)

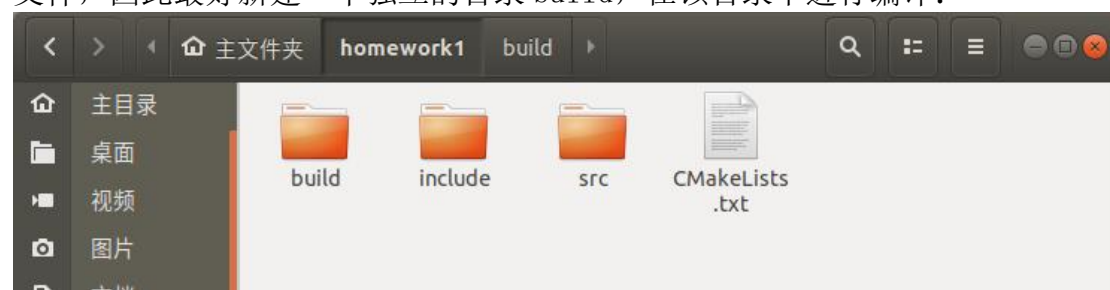
#5.set environment variable, 设置环境变量, 编译用到的源文件全部都要放到这
里, 否则编译能够通过, 但是执行的时候会出现各种问题, 比如"symbol lookup
error xxxxx , undefined symbol"
SET(TEST_MATH
${DIR_SRCS}
)|

#6.add executable file, 添加要编译的可执行文件
ADD_EXECUTABLE(${PROJECT_NAME} ${TEST_MATH})

#7.add link library, 添加可执行文件所需要的库, 比如我们用到了libm.so (命名
规则: lib+name+.so) , 就添加该库的名称
TARGET_LINK_LIBRARIES(${PROJECT_NAME} m)
```

3) 编译和运行程序

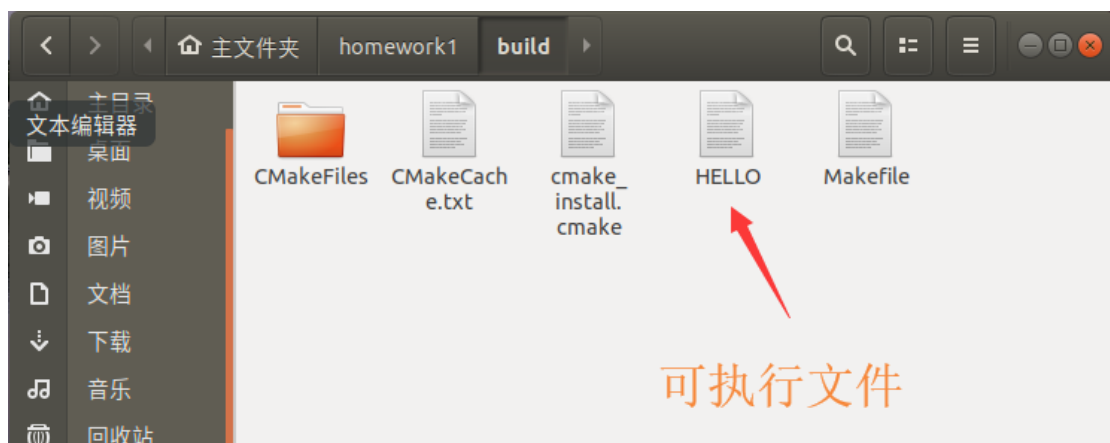
准备好了以上的所有材料，接下来，就可以编译了，由于编译中出现许多中间的文件，因此最好新建一个独立的目录 build，在该目录下进行编译：



编译步骤如下所示：

```
wyg@wyg-virtual-machine: ~/homework1/build
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)
(base) wyg@wyg-virtual-machine:~/homework1$ mkdir build
(base) wyg@wyg-virtual-machine:~/homework1$ cd build
(base) wyg@wyg-virtual-machine:~/homework1/build$ cmake ..
-- The C compiler identification is GNU 7.4.0
-- The CXX compiler identification is GNU 7.4.0
-- Check for working C compiler: /usr/bin/cc
-- Check for working C compiler: /usr/bin/cc -- works
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Detecting C compile features
-- Detecting C compile features - done
-- Check for working CXX compiler: /usr/bin/c++
-- Check for working CXX compiler: /usr/bin/c++ -- works
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Detecting CXX compile features
-- Detecting CXX compile features - done
-- Configuring done
-- Generating done
-- Build files have been written to: /home/wyg/homework1/build
(base) wyg@wyg-virtual-machine:~/homework1/build$ make
(base) wyg@wyg-virtual-machine:~/homework1/build$ make
-- Configuring done
-- Generating done
-- Build files have been written to: /home/wyg/homework1/build
Scanning dependencies of target HELLO
[ 33%] Building C object CMakeFiles/HELLO.dir/src/hello.c.o
[ 66%] Building C object CMakeFiles/HELLO.dir/src/useHello.c.o
[100%] Linking C executable HELLO
[100%] Built target HELLO
```

注意在 build 的目录下生成了一个可执行的文件 HELLO(忘记改成 sayhello 了)



运行获取结果如下:

```
hello/n(base) wyg@wyg-virtual-machine:~/homework1/build$ cmake ..
-- Configuring done
-- Generating done
-- Build files have been written to: /home/wyg/homework1/build
(base) wyg@wyg-virtual-machine:~/homework1/build$ make
Scanning dependencies of target HELLO
[ 33%] Building C object CMakeFiles/HELLO.dir/src/hello.c.o
[ 66%] Linking C executable HELLO
[100%] Built target HELLO
(base) wyg@wyg-virtual-machine:~/homework1/build$ ./HELLO
hello
(base) wyg@wyg-virtual-machine:~/homework1/build$
```

4) 最后执行 make install

刚开始失败了，于是找教程，需要重写 3 个 cmakeLists.txt 文件
在对应路径下新建文件

```
wyg@wyg-virtual-machine: ~/homework1/build
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)
(base) wyg@wyg-virtual-machine:~/homework1$ cd src
(base) wyg@wyg-virtual-machine:~/homework1/src$ touch CMakeLists.txt
(base) wyg@wyg-virtual-machine:~/homework1/src$ cd ..
(base) wyg@wyg-virtual-machine:~/homework1$ cd include
(base) wyg@wyg-virtual-machine:~/homework1/include$ touch CMakeLists.txt
(base) wyg@wyg-virtual-machine:~/homework1/include$ cd ..
(base) wyg@wyg-virtual-machine:~/homework1$ cd build
(base) wyg@wyg-virtual-machine:~/homework1/build$ cmake ..
CMake Error at src/CMakeLists.txt:7 (ADD_EXECUTABLE):
```

总目录下的 cmakeLists.txt 文件
需要添加子目录操作

```
打开(O)  CMakeLists.txt  保存(S)
~/homework1

#cmake version, 指定cmake版本
cmake_minimum_required(VERSION 3.2)

#project name, 指定项目的名称，一般和项目的文件夹名称对应
PROJECT(HELLO)

ADD_SUBDIRECTORY(include)
ADD_SUBDIRECTORY(src)

SET(CMAKE_BUILD_TYPE "Release")

#INSTALL(TARGETS libhello LIBRARY DESTINATION /usr/local/lib)
#INSTALL(/libhello/hello.h /usr/local/include)
```

include 目录下的 cmakeLists.txt 文件
设置了一些依赖关系

```
打开(O)  CMakeLists.txt  保存(S)
~/homework1/include

SET(LIB_SRC hello.c)

ADD_DEFINITIONS("-DLIBHELLO_BUILD")

ADD_LIBRARY(include SHARED ${LIB_SRC})

SET(LIBRARY_OUTPUT_PATH ${PROJECT_BINARY_DIR}/lib)

SET_TARGET_PROPERTIES(include PROPERTIES OUTPUT_NAME "sayHello")

INSTALL(TARGETS include LIBRARY DESTINATION /usr/local/lib)
INSTALL(FILES "hello.h" DESTINATION /usr/local/include)
```

src 目录下的 cmakeLists.txt 文件
也设置了一些依赖关系



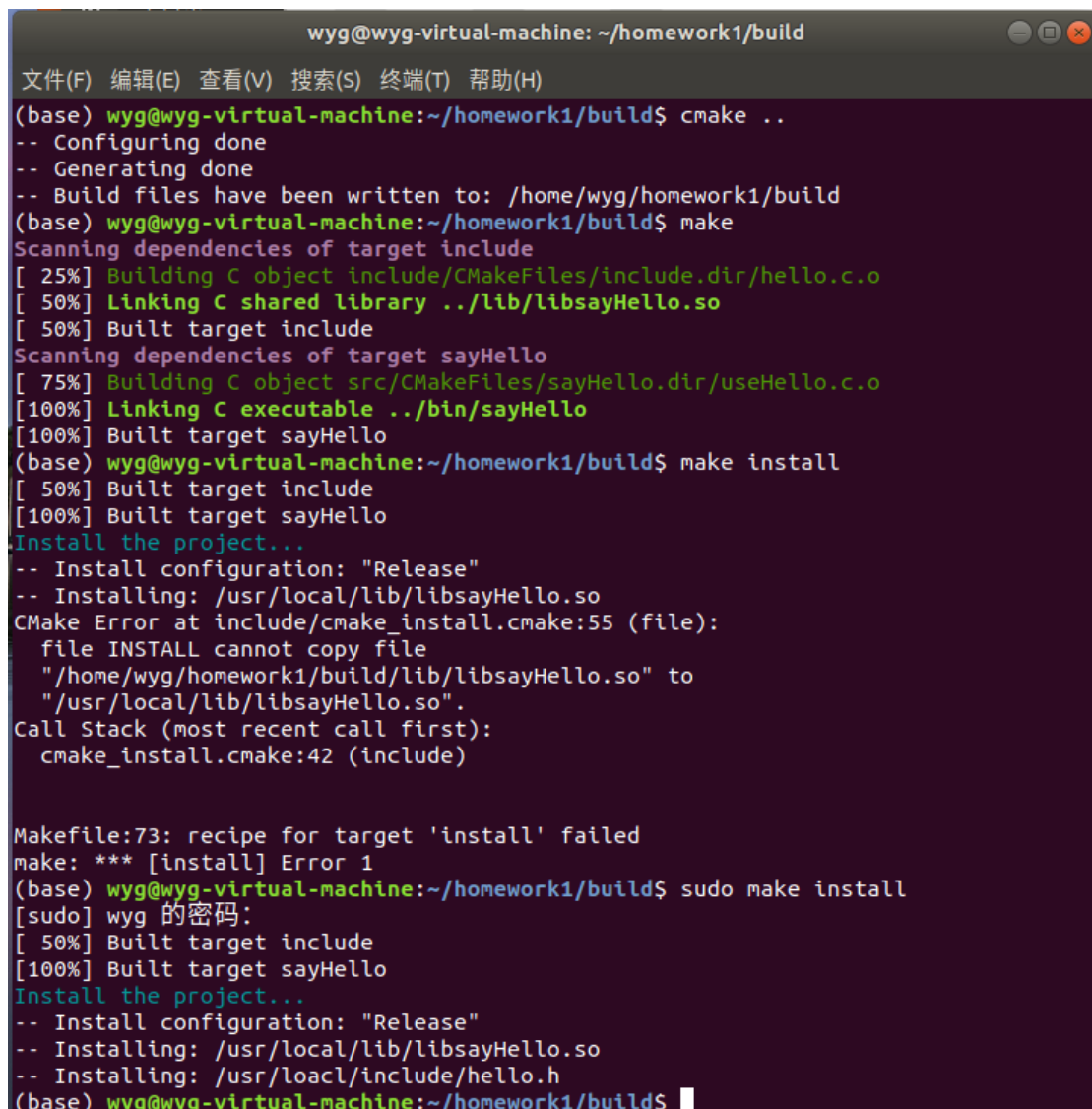
```
LINK_DIRECTORIES(${PROJECT_SOURCE_DIR})
INCLUDE_DIRECTORIES(${PROJECT_SOURCE_DIR}/include)

SET(APP_SRC useHello.c)
SET(EXECUTABLE_OUTPUT_PATH ${PROJECT_BINARY_DIR}/bin)

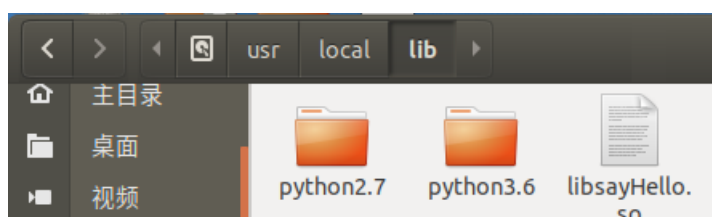
ADD_EXECUTABLE(sayHello ${APP_SRC})

TARGET_LINK_LIBRARIES(sayHello include)
```

编译并添加到库中

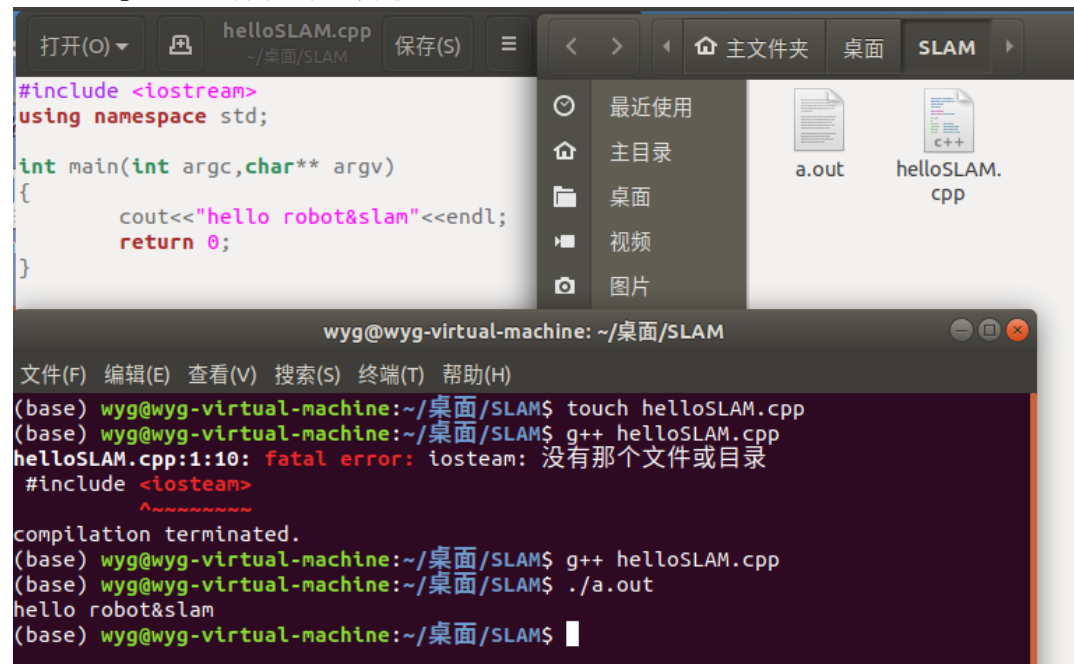


完成安装



附.书上示例代码练习

尝试用 g++ 单文件编译，并执行



The image shows a file manager window with the file `helloSLAM.cpp` open. The code is as follows:

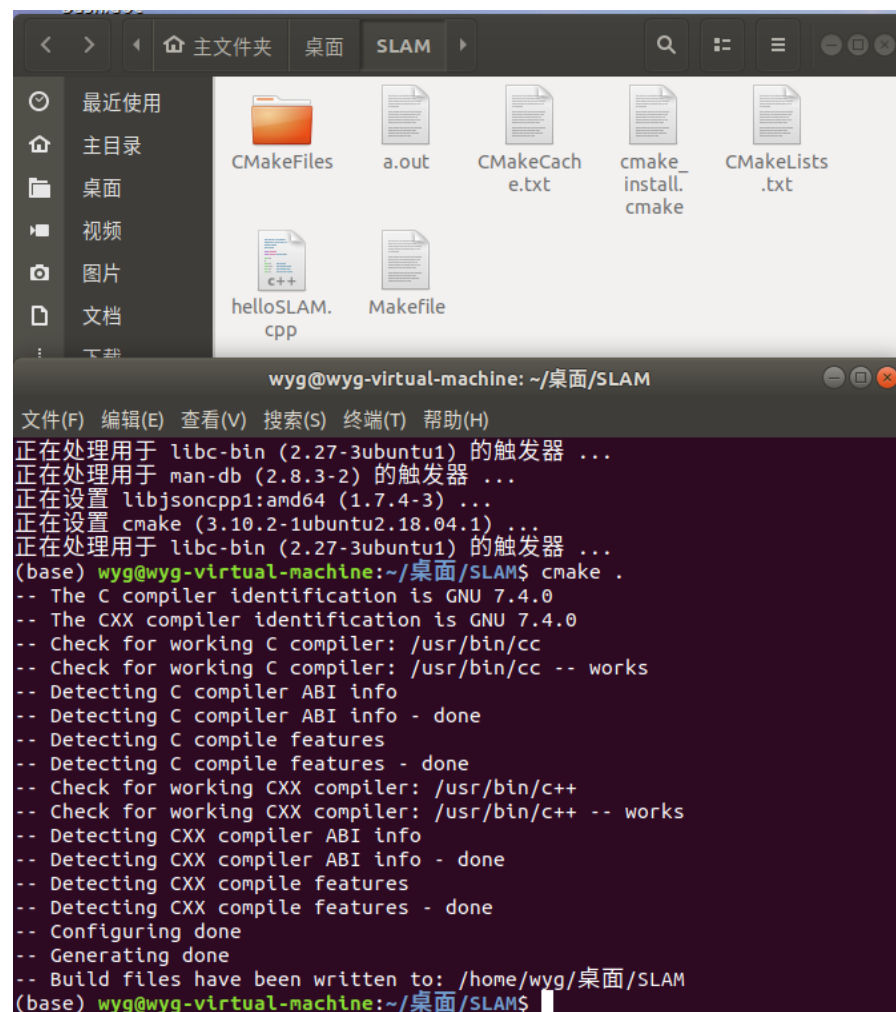
```
#include <iostream>
using namespace std;

int main(int argc, char** argv)
{
    cout << "hello robot&slam" << endl;
    return 0;
}
```

Below the file manager is a terminal window titled `wyg@wyg-virtual-machine: ~/桌面/SLAM`. The terminal shows the following commands and output:

```
(base) wyg@wyg-virtual-machine:~/桌面/SLAM$ touch helloSLAM.cpp
(base) wyg@wyg-virtual-machine:~/桌面/SLAM$ g++ helloSLAM.cpp
helloSLAM.cpp:1:10: fatal error: iostream: 没有那个文件或目录
#include <iostream>
         ^~~~~~
compilation terminated.
(base) wyg@wyg-virtual-machine:~/桌面/SLAM$ g++ helloSLAM.cpp
(base) wyg@wyg-virtual-machine:~/桌面/SLAM$ ./a.out
hello robot&slam
(base) wyg@wyg-virtual-machine:~/桌面/SLAM$
```

使用 cmake 编译

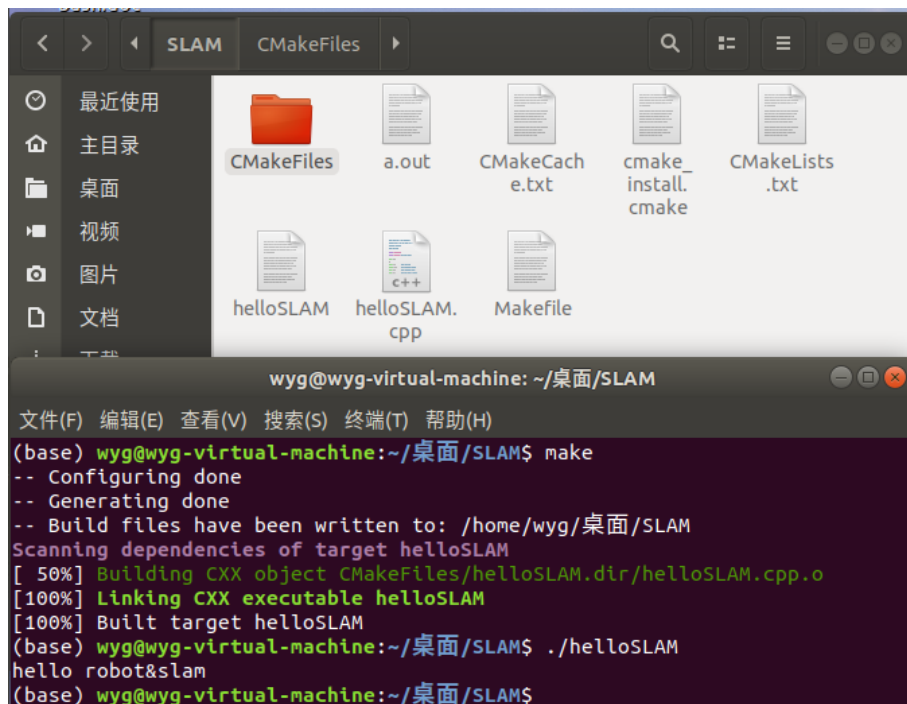


The image shows a file manager window with the file `helloSLAM.cpp` open. The file manager also shows other files like `CMakeFiles`, `a.out`, `CMakeCache.txt`, `cmake_install.cmake`, `CMakeLists.txt`, and `Makefile`.

Below the file manager is a terminal window titled `wyg@wyg-virtual-machine: ~/桌面/SLAM`. The terminal shows the following commands and output:

```
(base) wyg@wyg-virtual-machine:~/桌面/SLAM$ cmake .
-- The C compiler identification is GNU 7.4.0
-- The CXX compiler identification is GNU 7.4.0
-- Check for working C compiler: /usr/bin/cc
-- Check for working C compiler: /usr/bin/cc -- works
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Detecting C compile features
-- Detecting C compile features - done
-- Check for working CXX compiler: /usr/bin/c++
-- Check for working CXX compiler: /usr/bin/c++ -- works
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Detecting CXX compile features
-- Detecting CXX compile features - done
-- Configuring done
-- Generating done
-- Build files have been written to: /home/wyg/桌面/SLAM
(base) wyg@wyg-virtual-machine:~/桌面/SLAM$
```

生成了可执行文件



CMakeLists.txt 的写法



生成静态库：

The image shows a CMakeLists.txt file and a terminal window. The CMakeLists.txt file contains the following code:

```
//这是一个库文件
#include <iostream>
using namespace std;
void printHello()
{
    cout<<"Hello SLAM"<<endl;
}

# 声明要求的 cmake 最低版本
cmake_minimum_required( VERSION 2.8 )

# 声明一个 cmake 工程
project( HelloSLAM )

# 添加一个可执行程序
# 语法: add_executable( 程序名 源代码文件 )
add_executable( helloSLAM helloSLAM.cpp )
add_library( hello libHelloSLAM.cpp )
```

A red arrow points from the text "添加到CMakeLists" to the `add_library` line in the CMakeLists.txt file.

The terminal window shows the following commands and output:

```
wyg@wyg-virtual-machine: ~/桌面/SLAM
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)
(base) wyg@wyg-virtual-machine:~/桌面/SLAM$ touch libHelloSLAM.cpp
(base) wyg@wyg-virtual-machine:~/桌面/SLAM$ cd build
bash: cd: build: 没有那个文件或目录
(base) wyg@wyg-virtual-machine:~/桌面/SLAM$ mkdir build
(base) wyg@wyg-virtual-machine:~/桌面/SLAM$ cd build
(base) wyg@wyg-virtual-machine:~/桌面/SLAM/build$ cmake ..
-- Configuring done
-- Generating done
-- Build files have been written to: /home/wyg/桌面/SLAM
(base) wyg@wyg-virtual-machine:~/桌面/SLAM/build$ make
make: *** 没有指明目标并且找不到 makefile。 停止。
(base) wyg@wyg-virtual-machine:~/桌面/SLAM/build$ cd ..
(base) wyg@wyg-virtual-machine:~/桌面/SLAM$ make
[ 50%] Built target helloSLAM
Scanning dependencies of target hello
[ 75%] Building CXX object CMakeFiles/hello.dir/libHelloSLAM.cpp.o
[100%] Linking CXX static library libhello.a
[100%] Built target hello
(base) wyg@wyg-virtual-machine:~/桌面/SLAM$
```

The terminal output shows the successful creation of the static library `libhello.a`.

生成共享库:

The image illustrates the process of generating a shared library (libhello.so) from a CMake project. It is divided into three main sections:

1. CMakeLists.txt Configuration

```
//这是一个库文件
#include <iostream>
using namespace std;
void printHello()
{
    cout<<"Hello SLAM"<<endl;
}

# 声明要求的 cmake 最低版本
cmake_minimum_required( VERSION 2.8 )

# 声明一个 cmake 工程
project( HelloSLAM )

# 添加一个可执行程序
# 语法: add_executable( 程序名 源代码文件 )
add_executable( helloSLAM helloSLAM.cpp )
add_library( hello SHARED libHelloSLAM.cpp )
```

2. File Explorer View

The file explorer shows the contents of the `build` directory. A red arrow points from the `libhello.a` file to the `libhello.so` file, with the label "共享库" (Shared Library) in the center. The files listed are:

- build
- CMakeFiles
- a.out
- CMakeCache.txt
- cmake_install.cmake
- CMakeLists.txt
- helloSLAM
- helloSLAM.cpp
- libhello.a
- libhello.so

3. Terminal Output

```
wyg@wyg-virtual-machine: ~/桌面/SLAM

文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)

(base) wyg@wyg-virtual-machine:~/桌面/SLAM/build$ make
make: *** 没有指明目标并且找不到 makefile。 停止。
(base) wyg@wyg-virtual-machine:~/桌面/SLAM/build$ cd ..
(base) wyg@wyg-virtual-machine:~/桌面/SLAM$ make
[ 50%] Built target helloSLAM
Scanning dependencies of target hello
[ 75%] Building CXX object CMakeFiles/hello.dir/libHelloSLAM.cpp.o
[100%] Linking CXX static library libhello.a
[100%] Built target hello
(base) wyg@wyg-virtual-machine:~/桌面/SLAM$ cmake .
-- Configuring done
-- Generating done
-- Build files have been written to: /home/wyg/桌面/SLAM
(base) wyg@wyg-virtual-machine:~/桌面/SLAM$ make
[ 50%] Built target helloSLAM
[ 75%] Building CXX object CMakeFiles/hello.dir/libHelloSLAM.cpp.o
[100%] Linking CXX shared library libhello.so
[100%] Built target hello
(base) wyg@wyg-virtual-machine:~/桌面/SLAM$
```

从编译到执行：

The screenshot displays a Linux desktop environment with a sidebar on the left containing icons for Firefox, a mail client, a file manager, and a terminal. The main workspace is divided into three panels:

- Top Left Panel (CMakeLists.txt):** Contains CMake configuration code:

```
# 声明要求的 cmake 最低版本
cmake_minimum_required( VERSION 2.8 )

# 声明一个 cmake 工程
project( HelloSLAM )

# 添加一个可执行程序
# 语法: add_executable( 程序名 源代码文件 )
add_executable( helloSLAM helloSLAM.cpp )
add_library( hello_shared SHARED libHelloSLAM.cpp )
add_executable( useHello useHello.cpp )
target_link_libraries( useHello hello_shared )
```
- Top Right Panel (libHelloSLAM.h):** Contains the header file definition:

```
#ifndef LIBHELLOSLAM_H_
#define LIBHELLOSLAM_H_
void printHello();
#endif
```
- Bottom Panel (Terminal):** Shows the execution of the build process:

```
(base) wyg@wyg-virtual-machine: ~/桌面/SLAM$ cmake .
-- Configuring done
-- Generating done
-- Build files have been written to: /home/wyg/桌面/SLAM
(base) wyg@wyg-virtual-machine: ~/桌面/SLAM$ make
Scanning dependencies of target hello_shared
[ 16%] Building CXX object CMakeFiles/hello_shared.dir/libHelloSLAM.cpp.o
[ 33%] Linking CXX shared library libhello_shared.so
[ 33%] Built target hello_shared
Scanning dependencies of target useHello
[ 50%] Linking CXX executable useHello
[ 66%] Built target useHello
[100%] Built target helloSLAM
(base) wyg@wyg-virtual-machine: ~/桌面/SLAM$ ./useHello
Hello SLAM
(base) wyg@wyg-virtual-machine: ~/桌面/SLAM$
```

The **File Manager** window at the bottom shows the contents of the `~/桌面/SLAM` directory. It lists various files and folders created during the build process, including `build`, `CMakeFiles`, `a.out`, `CMakeCache.txt`, `cmake_install.cmake`, `CMakeLists.txt`, `helloSLAM`, `helloSLAM.cpp`, `libhello.a`, `libhello.so`, `libhello_shared.so`, `libHelloSLAM.h`, `libHelloSLAM.cpp`, `Makefile`, `useHello`, and `useHello.cpp`. Red boxes highlight `libhello_shared.so` and `useHello`, with an arrow pointing from the text "共享库文件 可执行文件" (Shared library file, Executable file) to these two files.

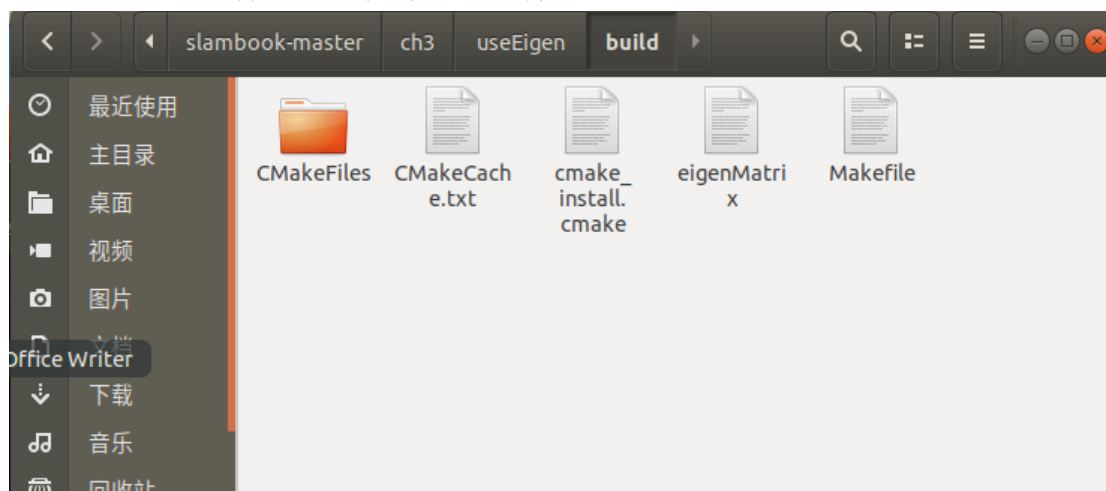
❖ 运行参考书《视觉 SLAM 十四讲》P45-47 的

eigenMatrix.cpp 程序并熟悉代码;

1) 编译

```
wyg@wyg-virtual-machine: ~/桌面/slambook-master/ch3/useEigen/build
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)
(base) wyg@wyg-virtual-machine:~/桌面/slambook-master/ch3/useEigen$ mkdir build
(base) wyg@wyg-virtual-machine:~/桌面/slambook-master/ch3/useEigen$ cd build
(base) wyg@wyg-virtual-machine:~/桌面/slambook-master/ch3/useEigen/build$ cmake ..
-- The C compiler identification is GNU 7.4.0
-- The CXX compiler identification is GNU 7.4.0
-- Check for working C compiler: /usr/bin/cc
-- Check for working C compiler: /usr/bin/cc -- works
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Detecting C compile features
-- Detecting C compile features - done
-- Check for working CXX compiler: /usr/bin/c++
-- Check for working CXX compiler: /usr/bin/c++ -- works
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Detecting CXX compile features
-- Detecting CXX compile features - done
-- Configuring done
-- Generating done
-- Build files have been written to: /home/wyg/桌面/slambook-master/ch3/useEigen/build
(base) wyg@wyg-virtual-machine:~/桌面/slambook-master/ch3/useEigen/build$ make
Scanning dependencies of target eigenMatrix
[ 50%] Building CXX object CMakeFiles/eigenMatrix.dir/eigenMatrix.cpp.o
[100%] Linking CXX executable eigenMatrix
[100%] Built target eigenMatrix
```

创建了 build 文件，避免编译后的文件太乱



查看 CMakeLists.txt 包含了 eigen3 线性代数库

```
CMakeLists.txt
~/桌面/slambook-mas... 保存(S)

cmake_minimum_required( VERSION 2.8 )
project( useEigen )

set( CMAKE_BUILD_TYPE "Release" )
set( CMAKE_CXX_FLAGS "-O3" )

# 添加Eigen头文件
include_directories( "/usr/include/eigen3" )

# in osx and brew install
# include_directories( /usr/local/Cellar/eigen/
3.3.3/include/eigen3 )

CMake 制表符宽度: 8 第 1 行, 第 1 列 插入
```

2) 运行

查看输出结果

```
wyg@wyg-virtual-machine: ~/桌面/slambook-master/ch3/useEigen/build
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)
(base) wyg@wyg-virtual-machine:~/桌面/slambook-master/ch3/useEigen/build$ ./eigenMatrix
1 2 3
4 5 6
1      2      3
4      5      6
10
28
32
77
0.680375  0.59688 -0.329554
-0.211234 0.823295 0.536459
0.566198 -0.604897 -0.444451

0.680375 -0.211234 0.566198
0.59688  0.823295 -0.604897
-0.329554 0.536459 -0.444451
1.61307
1.05922
6.80375  5.9688 -3.29554
-2.11234 8.23295 5.36459
5.66198 -6.04897 -4.44451
-0.198521 2.22739 2.8357
1.00605 -0.555135 -1.41603
-1.62213 3.59308 3.28973
0.208598
Eigen values =
0.0242899
0.992154
1.80558
Eigen vectors =
-0.549013 -0.735943 0.396198
0.253452 -0.598296 -0.760134
-0.796459 0.316906 -0.514998
time use in normal inverse is 0.106ms
time use in Qr decomposition is 0.054ms
(base) wyg@wyg-virtual-machine:~/桌面/slambook-master/ch3/useEigen/build$
```

查看 eigenMatrix.cpp 程序复习

```
#include <iostream>
using namespace std;
#include <ctime>
// Eigen 部分
```

```

#include <Eigen/Core>
// 稠密矩阵的代数运算（逆，特征值等）
#include <Eigen/Dense>

#define MATRIX_SIZE 50

/*****
* 本程序演示了 Eigen 基本类型的使用
*****/

int main( int argc, char** argv )
{
    // Eigen 中所有向量和矩阵都是 Eigen::Matrix，它是一个模板类。它的前三个参数为：数据类型，行，列
    // 声明一个 2*3 的 float 矩阵
    Eigen::Matrix<float, 2, 3> matrix_23;

    // 同时，Eigen 通过 typedef 提供了许多内置类型，不过底层仍是 Eigen::Matrix
    // 例如 Vector3d 实质上是 Eigen::Matrix<double, 3, 1>，即三维向量
    Eigen::Vector3d v_3d;
    // 这是一样的
    Eigen::Matrix<float,3,1> vd_3d;

    // Matrix3d 实质上是 Eigen::Matrix<double, 3, 3>
    Eigen::Matrix3d matrix_33 = Eigen::Matrix3d::Zero(); //初始化为零
    // 如果不确定矩阵大小，可以使用动态大小的矩阵
    Eigen::Matrix< double, Eigen::Dynamic, Eigen::Dynamic > matrix_dynamic;

    // 更简单的
    Eigen::MatrixXf matrix_x;
    // 这种类型还有很多，我们不一一列举

    // 下面是对 Eigen 阵的操作
    // 输入数据（初始化）
    matrix_23 << 1, 2, 3, 4, 5, 6;
    // 输出
    cout << matrix_23 << endl;

    // 用()访问矩阵中的元素
    for (int i=0; i<2; i++) {
        for (int j=0; j<3; j++)
            cout<<matrix_23(i,j)<<"\t";
        cout<<endl;
    }
}

```

```

}

// 矩阵和向量相乘（实际上仍是矩阵和矩阵）
v_3d << 3, 2, 1;
vd_3d << 4,5,6;
// 但是在 Eigen 里你不能混合两种不同类型的矩阵，像这样是错的
// Eigen::Matrix<double, 2, 1> result_wrong_type = matrix_23 * v_3d
;

// 应该显式转换
Eigen::Matrix<double, 2, 1> result = matrix_23.cast<double>() * v_3d;

cout << result << endl;

Eigen::Matrix<float, 2, 1> result2 = matrix_23 * vd_3d;
cout << result2 << endl;

// 同样你不能搞错矩阵的维度
// 试着取消下面的注释，看看 Eigen 会报什么错
// Eigen::Matrix<double, 2, 3> result_wrong_dimension = matrix_23.cast<double>() * v_3d;

// 一些矩阵运算
// 四则运算就不演示了，直接用+-* /即可。
matrix_33 = Eigen::Matrix3d::Random(); // 随机数矩阵
cout << matrix_33 << endl << endl;

cout << matrix_33.transpose() << endl; // 转置
cout << matrix_33.sum() << endl; // 各元素和
cout << matrix_33.trace() << endl; // 迹
cout << 10*matrix_33 << endl; // 数乘
cout << matrix_33.inverse() << endl; // 逆
cout << matrix_33.determinant() << endl; // 行列式

// 特征值
// 实对称矩阵可以保证对角化成功
Eigen::SelfAdjointEigenSolver<Eigen::Matrix3d> eigen_solver ( matrix_33.transpose()*matrix_33 );
cout << "Eigen values = \n" << eigen_solver.eigenvalues() << endl;
cout << "Eigen vectors = \n" << eigen_solver.eigenvectors() << endl
;

// 解方程
// 我们求解 matrix_NN * x = v_Nd 这个方程
// N 的大小在前边的宏里定义，它由随机数生成

```

```

// 直接求逆自然是最直接的，但是求逆运算量大

Eigen::Matrix< double, MATRIX_SIZE, MATRIX_SIZE > matrix_NN;
matrix_NN = Eigen::MatrixXd::Random( MATRIX_SIZE, MATRIX_SIZE );
Eigen::Matrix< double, MATRIX_SIZE, 1> v_Nd;
v_Nd = Eigen::MatrixXd::Random( MATRIX_SIZE,1 );

clock_t time_stt = clock(); // 计时
// 直接求逆
Eigen::Matrix<double,MATRIX_SIZE,1> x = matrix_NN.inverse()*v_Nd;
cout <<"time use in normal inverse is " << 1000* (clock() - time_stt)/((double)CLOCKS_PER_SEC << "ms"<< endl;

// 通常用矩阵分解来求，例如 QR 分解，速度会快很多
time_stt = clock();
x = matrix_NN.colPivHouseholderQr().solve(v_Nd);
cout <<"time use in Qr decomposition is " <<1000* (clock() - time_stt)/((double)CLOCKS_PER_SEC <<"ms" << endl;

return 0;
}

```