

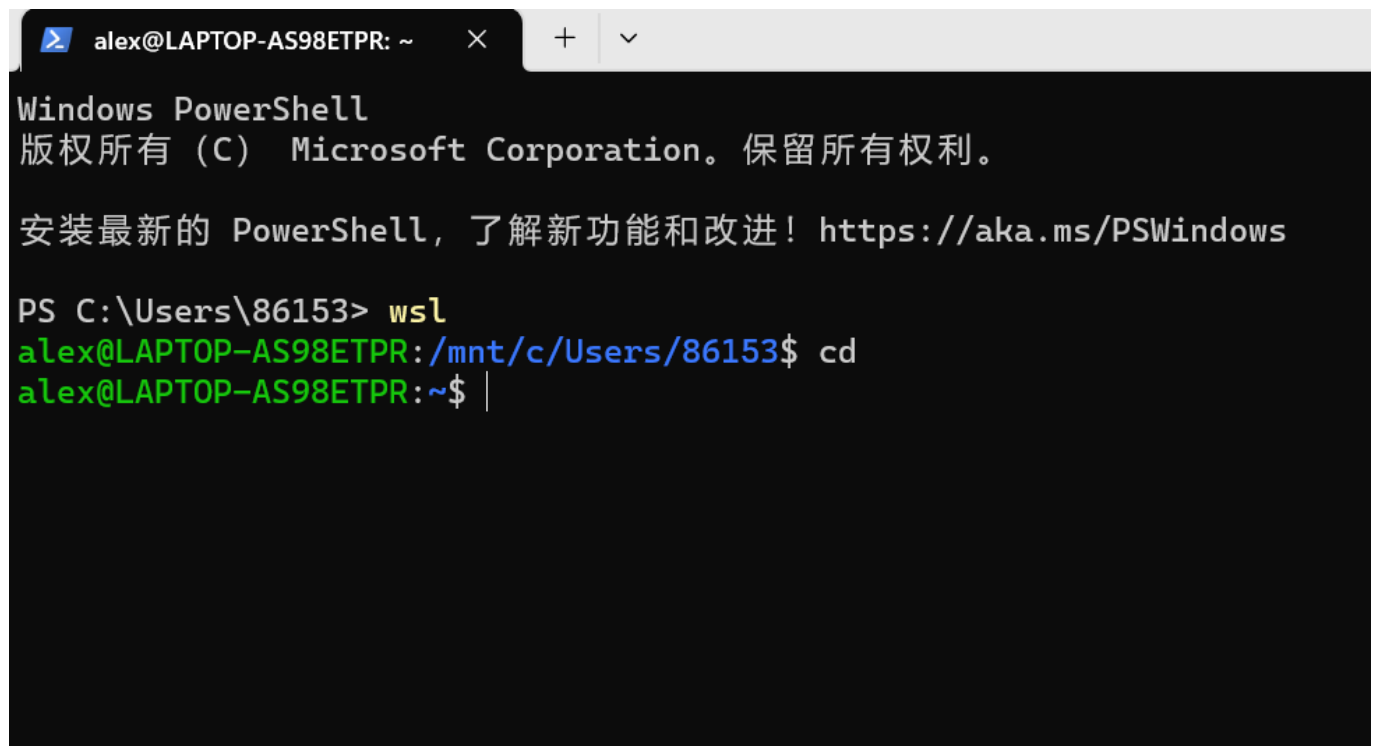
# 任务一 linux系统

任务要求：可以参考上述资料，在自己的机器上配置双系统或者使用WSL2，拥有自己的linux系统

第一次装虚拟机，而且准备开始任务时没有U盘，时间宝贵，就先装了WSL将就一下，而且考虑到WSL虽然也可以相对方便的访问大部分windows系统上的文件，但没有可视化界面，所以决定在后续完成本周全部任务后安装双系统。

**安装过程**基本参照[如何在Windows11上安装WSL2的Ubuntu22.04（包括换源）\\_wsl2换源\\_syqkali的博客-CSDN博客](#)，没有遇到困难

结果展示



```
alex@LAPTOP-AS98ETPR: ~
Windows PowerShell
版权所有 (C) Microsoft Corporation。保留所有权利。

安装最新的 PowerShell，了解新功能和改进！ https://aka.ms/PSWindows

PS C:\Users\86153> wsl
alex@LAPTOP-AS98ETPR:/mnt/c/Users/86153$ cd
alex@LAPTOP-AS98ETPR:~$ |
```

补充说明：目前已经再次安装了双系统，并同时配置了Opencv库和VScode。

## 任务三：linux命令

任务要求：参考上述内容，使用命令行在 主目录 也就是 ~ 下，使用mkdir新建一个任务文件夹，使用cd进入该文件夹，使用touch新建cpp代码文件，编写helloworld代码，使用gcc编译，执行可执行文件，能够正确输出。（能够自己查阅不熟悉的命令）

Linux的一大特色就是命令行操作，通过查阅博客，我熟悉了

`cd`ls`mkdir`touch`tree`file`cat`rm`find`help`等常用操作，学习命令行如同学习一门语言，有很多细节是需要不断熟悉才能真正掌握的。

结果

创建了相应的Dir\_1文件夹，并使用touch创建了cpp代码

```
alex@LAPTOP-AS98ETPR:~/Dir_1$ ls -l
CMakeLists.txt
README.md
build
hello
hello.cpp
```

使用nano编辑代码

```
GNU nano 6.2 hello.cpp *
#include<iostream>
using namespace std;

int main()
{
    cout << "Hello World!"<<endl;
    return 0;
}
```

使用gcc编译，执行可执行文件，能够正确输出。

```
alex@LAPTOP-AS98ETPR:~/Dir_1$ nano hello.cpp
alex@LAPTOP-AS98ETPR:~/Dir_1$ g++ hello.cpp -o hello
alex@LAPTOP-AS98ETPR:~/Dir_1$ ./hello
Hello World!
alex@LAPTOP-AS98ETPR:~/Dir_1$ |
```

## 任务四：CMake简单实践

任务要求：参考上述资料，使用CMake编译任务三中的helloworld代码，然后cd进入build目录，使用cmake .. 命令，然后使用make命令生成执行文件

### 创建相应文件结构

```
| -CMakeLists.txt
| -helloworld.cpp
| -helloworld          # 任务三中使用gcc编译的生成的helloworld可执行文件
| -build
| -README.md          # 存放工程的说明文档
```

如图所示:

```
alex@LAPTOP-AS98ETPR:~/Dir_1$ ls -l
CMakeLists.txt
README.md
build
hello
hello.cpp
```

使用CMake编译上述代码，生成可执行文件

其中我的CMakeLists.txt是最简版本

```
GNU nano 6.2 CMakeLists.txt
cmake_minimum_required(VERSION 3.10)
project(MyProject)
set(CMAKE_CXX_STANDARD 11)
add_executable(hello hello.cpp)
```

编译操作如下:

```
alex@LAPTOP-AS98ETPR:~/Dir_1$ cd build
alex@LAPTOP-AS98ETPR:~/Dir_1/build$ cmake ..
-- Configuring done
-- Generating done
-- Build files have been written to: /home/alex/Dir_1/build
alex@LAPTOP-AS98ETPR:~/Dir_1/build$ make
Consolidate compiler generated dependencies of target hello
[ 50%] Building CXX object CMakeFiles/hello.dir/hello.cpp.o
[100%] Linking CXX executable hello
[100%] Built target hello
alex@LAPTOP-AS98ETPR:~/Dir_1/build$ ./hello
Hello World!
alex@LAPTOP-AS98ETPR:~/Dir_1/build$ |
```

## 任务六：CMake实践

任务要求：查阅CMake资料，能够在build文件夹下成功编译并运行可执行文件，不报错

之前我已经成功安装了opencv4,版本信息如下:

```
alex@LAPTOP-AS98ETPR:~/Dir_1$ pkg-config --modversion opencv4
4.5.4
alex@LAPTOP-AS98ETPR:~/Dir_1$ |
```

首先下载群文件，在Ubuntu上打开相应文件夹

```
PS C:\Users\86153\Desktop\RM\L_1\segment\segment> wsl
alex@LAPTOP-AS98ETPR:/mnt/c/Users/86153/Desktop/RM/L_1/segment/segment$ ls -l
CMakeLists.txt
build
contours.png
include
src
test_img.jpg
alex@LAPTOP-AS98ETPR:/mnt/c/Users/86153/Desktop/RM/L_1/segment/segment$ |
```

先对文件源码的图片读取地址进行稍微修正，之前我没有调整程序会报错之后补充CMakeLists.txt文件，如图所示

```
GNU nano 6.2 CMakeLists.txt
cmake_minimum_required(VERSION 2.8)
project(segmentation)
set(CMAKE_CXX_STANDARD 11)

# 查找OpenCV库
find_package(OpenCV REQUIRED)

# 添加OpenCV头文件的路径
include_directories(${OpenCV_INCLUDE_DIRS})

# 设置需要编译的源文件
set(SOURCE_FILES src/main.cpp src/segment.cpp include/segment.h)

# 创建可执行文件
add_executable(segmentation ${SOURCE_FILES})

# 链接OpenCV库
target_link_libraries(segmentation ${OpenCV_LIBS})
```

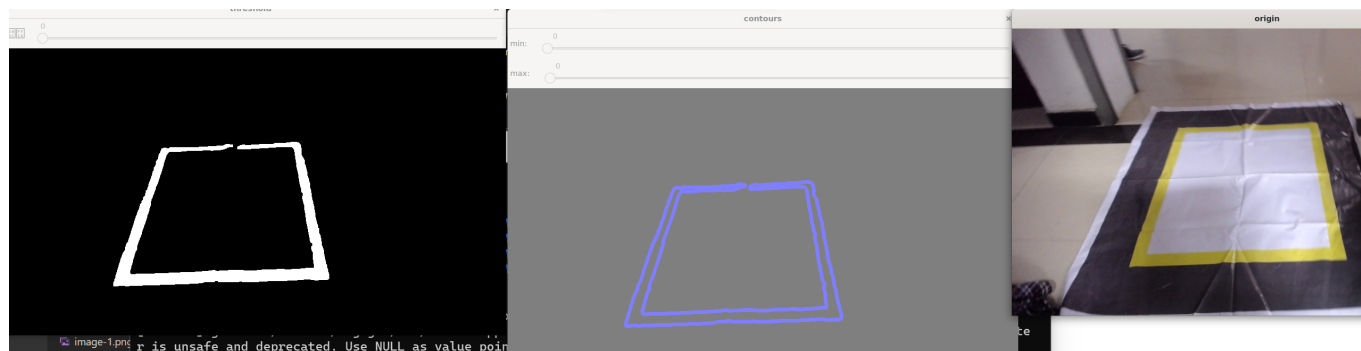
用Cmake对源代码进行编译并运行

```
alex@LAPTOP-AS98ETPR:/mnt/c/Users/86153/Desktop/RM/L_1/segment/segment$ nano CMakeLists.txt
alex@LAPTOP-AS98ETPR:/mnt/c/Users/86153/Desktop/RM/L_1/segment/segment$ cd build
alex@LAPTOP-AS98ETPR:/mnt/c/Users/86153/Desktop/RM/L_1/segment/segment/build$ cmake ..
CMake Deprecation Warning at CMakeLists.txt:1 (cmake_minimum_required):
  Compatibility with CMake < 2.8.12 will be removed from a future version of
  CMake.

  Update the VERSION argument <min> value or use a ...<max> suffix to tell
  CMake that the project does not need compatibility with older versions.

-- Configuring done
-- Generating done
-- Build files have been written to: /mnt/c/Users/86153/Desktop/RM/L_1/segment/segment/build
alex@LAPTOP-AS98ETPR:/mnt/c/Users/86153/Desktop/RM/L_1/segment/segment/build$ make
Consolidate compiler generated dependencies of target segmentation
[100%] Built target segmentation
alex@LAPTOP-AS98ETPR:/mnt/c/Users/86153/Desktop/RM/L_1/segment/segment/build$ ./segmentation
Original Image Size: [1920 x 1080]
```

运行结果如下



文件整体结构如图所示

```
PS C:\Users\86153\Desktop\RM\L_1\segment\segment> wsl
alex@LAPTOP-AS98ETPR:/mnt/c/Users/86153/Desktop/RM/L_1/segment/segment$ tree
.
├── CMakeLists.txt
├── bin
├── CMakeCache.txt
├── CMakeFiles
│   ├── 1.22.3
│   │   ├── CMakeCCompiler.cmake
│   │   ├── CMakeCXXCompiler.cmake
│   │   ├── CMakeDetermineCompilerABI_C.bin
│   │   ├── CMakeDetermineCompilerABI_CXX.bin
│   │   ├── CMakeSystem.cmake
│   │   ├── CompilerIdC
│   │   │   ├── CMakeCCompilerId.c
│   │   │   ├── a.out
│   │   │   └── log
│   │   └── CompilerIdCXX
│   │       ├── CMakeCXXCompilerId.cpp
│   │       ├── a.out
│   │       └── log
│   ├── CMakeDirectoryInformation.cmake
│   ├── CMakeOutput.log
│   ├── CMakeRules
│   ├── Makefile.cmake
│   ├── Makefile2
│   ├── TargetDirectories.txt
│   ├── cmake.check_cache
│   ├── progress.marks
│   └── segmentation.dir
│       ├── DependInfo.cmake
│       ├── build.make
│       ├── cmake_clean.cmake
│       ├── compiler_depend.internal
│       ├── compiler_depend.make
│       ├── compiler_depend.ts
│       ├── depend.make
│       ├── flags.make
│       ├── link.txt
│       ├── progress.make
│       └── run
│           ├── main.cpp.o
│           ├── main.cpp.o.d
│           ├── segment.cpp.o
│           └── segment.cpp.o.d
├── Makefile
├── cmake_install.cmake
├── segmentation
├── contours.png
├── include
│   └── segment.h
├── src
│   ├── main.cpp
│   └── segment.cpp
└── test_img.jpg

12 directories, 40 files
alex@LAPTOP-AS98ETPR:/mnt/c/Users/86153/Desktop/RM/L_1/segment/segment$ |
```