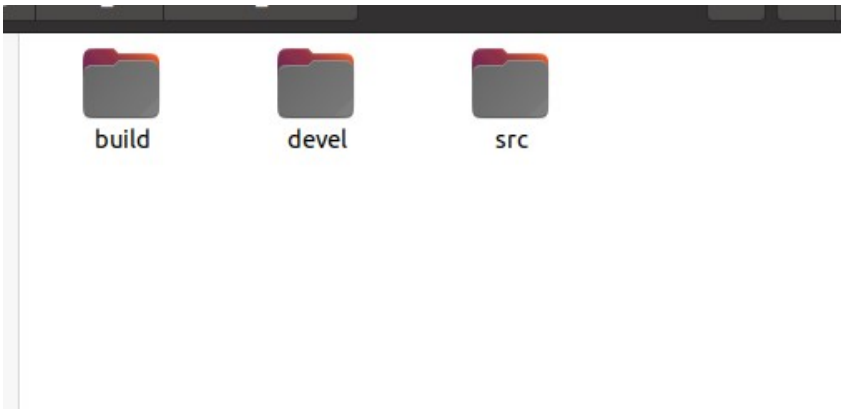
创建功能空间　依赖的包
Mkdir　文件夹名 创建一个文件夹　　TAB 键自动补齐

1、mkdir -p demo_01/src 创建了一个文件夹，这个文件家 demo_01
.并在下面方一个 src 为文件

2、cd demo_01　进入 demo_01 文件夹　这个文件家就算我们的项目的大文件家
在这个路径下 使用 catkin_make 命令
3、catkin_make
创建 ros 的一些基础文件

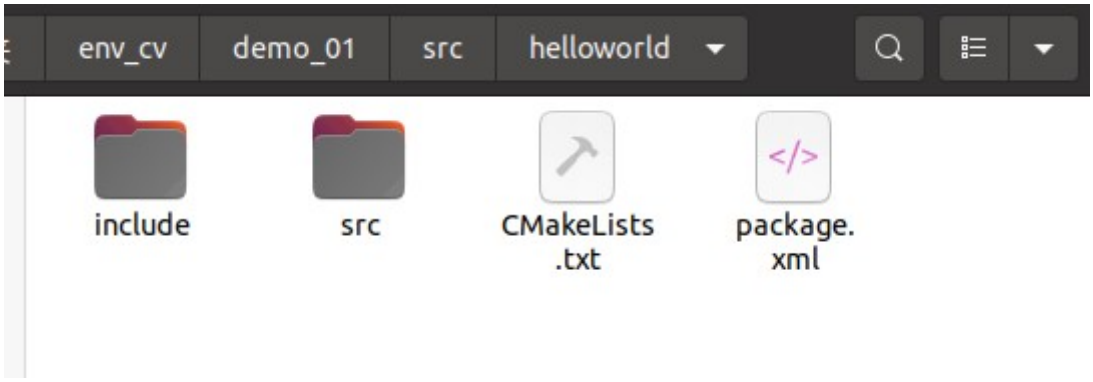
```
qinghuan@qinghuan-System-Product-Name:~/env_cv/demo_01$ catkin_make
Base path: /home/qinghuan/env_cv/demo_01
Source space: /home/qinghuan/env_cv/demo_01/src
Build space: /home/qinghuan/env_cv/demo_01/build
Devel space: /home/qinghuan/env_cv/demo_01/devel
Install space: /home/qinghuan/env_cv/demo_01/install
Creating symlink "/home/qinghuan/env_cv/demo_01/src/CMakeLists.txt" poi
"/opt/ros/noetic/share/catkin/cmake/toplevel.cmake"
####
#### Running command: "cmake /home/qinghuan/env_cv/demo_01/src -DCATKIN
EFIX=/home/qinghuan/env_cv/demo_01/devel -DCMAKE_INSTALL_PREFIX=/home/q
nv_cv/demo_01/install -G Unix Makefiles" in "/home/qinghuan/env_cv/demo
"
####
-- The C compiler identification is GNU 9.4.0
-- The CXX compiler identification is GNU 9.4.0
-- 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
```

这个命令会帮助我们创建 ros 的一些环境基础配置文件，出现了 build 文件家和 devel 文件夹 这几个和在后续需要继续理解使用



4、cd src 进入 src 文件夹
catkin_create_pkg ros 包的名字　roscpp rospy std_msgs
5、catkin_create_pkg hellowworld roscpp rospy std_msgs
这个是进入 src 文件 创建 ros 包　并添加依赖文件
依赖 rospp　rospy std_msgs 三个库
roscpp 是 c++文件　rospy 是 python 文件库　std_msgs 是标准消息库

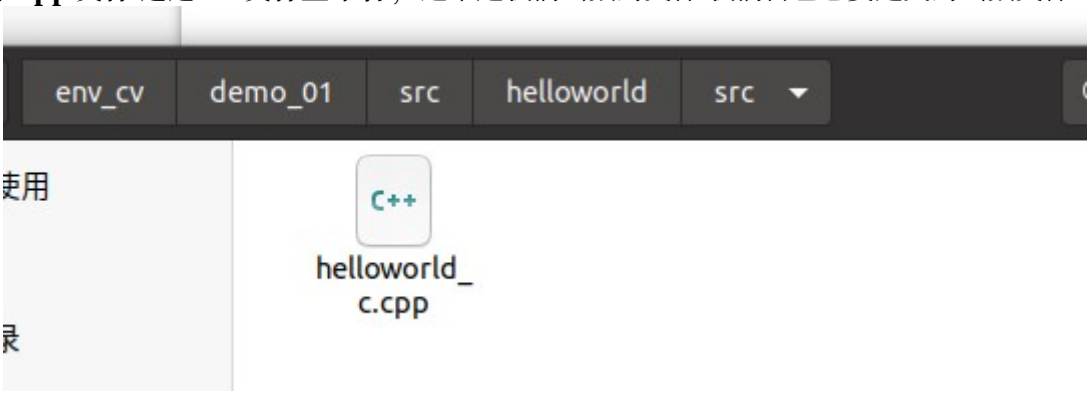
ros 包的名字 是我们自己创建文件的名字
helloworld 是 ros 包的名字 cmakelist 我看有人说是配置环境用的，就算需要那些依赖的文件
helloworld 文件下的四个文件如图所示



src 就算我们自己写程序的文件夹

6、写自己的 CPP 文件

创建一个 **cpp 文件 通过 txt 文件重命名**，这个是我们函数的文件 我们自己想要定义的函数文件



在写 ros 的文件的时候

1、包含 ros 的头文件 #include"ros/ros.h"

2、编写 main 函数

```
int main(int args , char * argv[])
{
}
    初始化节点 ros::init(args,argv,"hello_world");
);
输出日志 ROS_INFO(" hello world");
```

什么是节点：ros 是一和进程 也称之为 nodes 的分布式矿监 不同注解系统工作 从而分散计算压力 使用功能封装 主要指知道 IO 就可以复用 模块间的松耦合

打开(O) helloworld_c.cpp ~/env_cv/demo_01/src/helloworld/src 保存(S)

```cpp
#include"ros/ros.h"
//包含ros的头函数文件
//编写main 函数

//参赛 argc argv是字符串数组
int main(int argc, char *argv[])
{
        //初始化ros节点 init 传的擦丝农户 argc argv  hello_world是节点名称 字符串自己定义
        ros::init(argc,argv,"hello_wold");
        //ROS_INFO 日志输出
        ROS_INFO("hello world");


        return 0;
}
```

7、配置文件的依赖

136 和 149

在 src 文件夹同级的地方 会有 CMakeLists.txt 文件

136 行去掉注释 之后区执行这个 cpp 文件

add_executable cpp 文件 是我们想要执行的 main 函数文件 因此 这个地方要 **确认执行文件的名字** 的${PROJECT_NAME}_node_src 这个是这个文件的名字 通过前面这个映射到文件

```
## The recommended prefix ensures that target names across packages don't collide
add_executable(${PROJECT_NAME}_node src/helloworld_c.cpp)

## Rename C++ executable without prefix
## The above recommended prefix causes long target names, the following renames the
## target back to the shorter version for ease of user use
## e.g. "rosrun someones_pkg node" instead of "rosrun someones_pkg someones_pkg_nod
# set_target_properties(${PROJECT_NAME}_node PROPERTIES OUTPUT_NAME node PREFIX "")

## Add cmake target dependencies of the executable
## same as for the library above
# add_dependencies(${PROJECT_NAME}_node ${${PROJECT_NAME}_EXPORTED_TARGETS} $
```

目标依赖库 我认为这个地方大概率是 第三方库函数的东西

```
  ${catkin_EXPORTED_TARGETS})

## Specify libraries to link a library or executable target against
target_link_libraries(${PROJECT_NAME}_node
   ${catkin_LIBRARIES}
 )

#############
## Install ##
#############

# all install targets should use catkin DESTINATION variables
# See http://ros.org/doc/api/catkin/html/adv_user_guide/variables.html
```

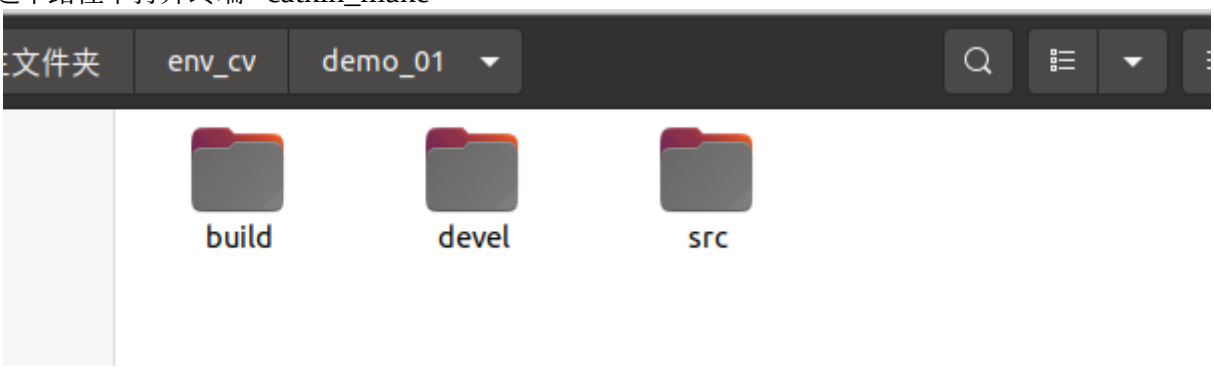
136 和 149 的地方要一致

```
136 add_executable(${PROJECT_NAME}_node src/helloworld_c.cpp)
137
138 ## Rename C++ executable without prefix
139 ## The above recommended prefix causes long target names, the following renames the
140 ## target back to the shorter version for ease of user use
141 ## e.g. "rosrun someones_pkg node" instead of "rosrun someones_pkg someones_pkg_node"
142 # set_target_properties(${PROJECT_NAME}_node PROPERTIES OUTPUT_NAME node PREFIX "")
143
144 ## Add cmake target dependencies of the executable
145 ## same as for the library above
146 # add_dependencies(${PROJECT_NAME}_node ${${PROJECT_NAME}_EXPORTED_TARGETS} $
    {catkin_EXPORTED_TARGETS})
147
148 ## Specify libraries to link a library or executable target against
149 target_link_libraries(${PROJECT_NAME}_node
150   ${catkin_LIBRARIES}
151 )
152
153 #############
```

8、在 demo_01 的 路径的终端 catkin_make 生成 ros 的执行文件

在 demo_01 的路径下 而不是 src 这个文件路径下

在这个路径下打开终端 catkin_make



成功的标志

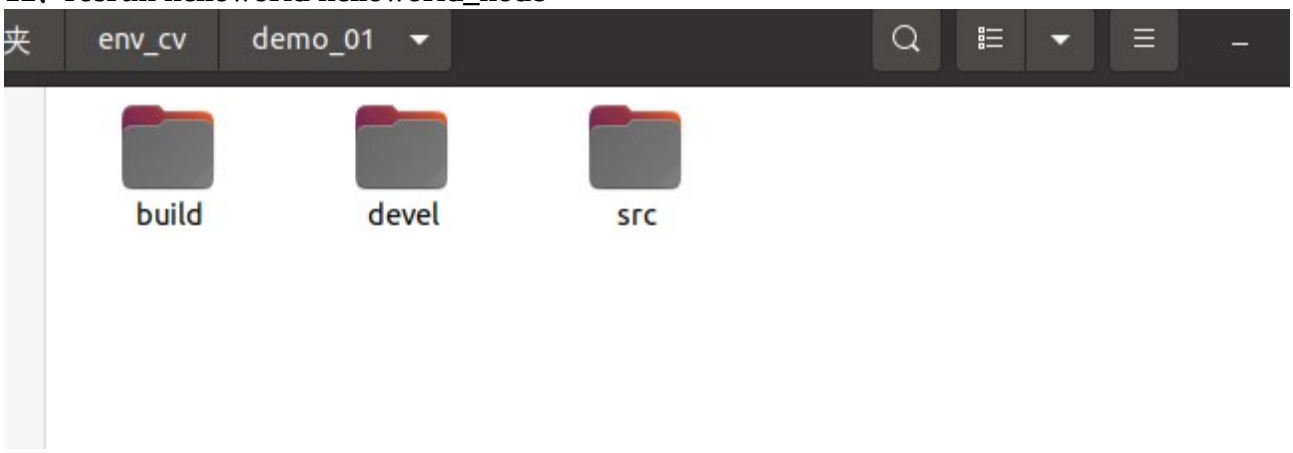
9、在一个新的 终端启动 roscore

打开 ros 这个功能 roscore

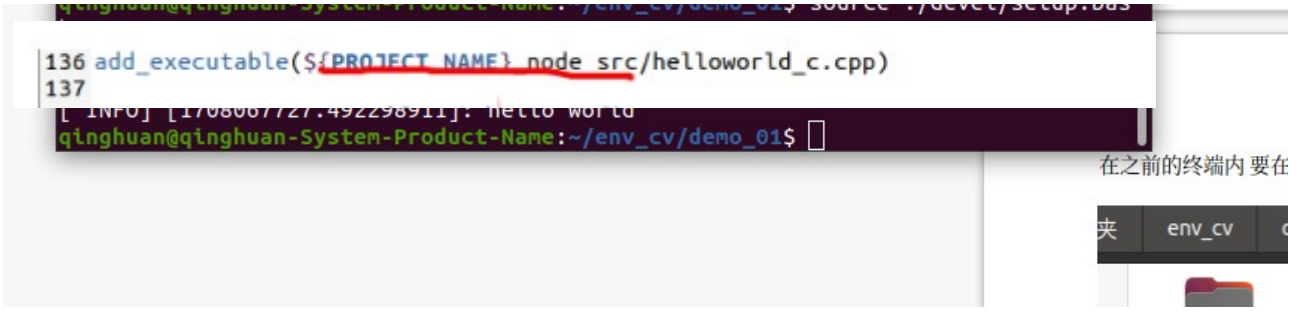
```
qinghuan@qinghuan-System-Product-Name:~$ roscore
... logging to /home/qinghuan/.ros/log/8a614e0a-cc9a-11ee-bbb7-1f2956
aunch-qinghuan-System-Product-Name-8516.log
Checking log directory for disk usage. This may take a while.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://qinghuan-System-Product-Name:39511/
ros_comm version 1.16.0


SUMMARY
========

PARAMETERS
 * /rosdistro: noetic
 * /rosversion: 1.16.0
```

roscore

10、在 demo_01 这个路径的终端下 运行两行命令
11、source ./devel/setup.bash
12、rosrun helloworld helloworld_node



分为三个部分 rosrun 运行一个文件 这个项目是 helloworld 这个 src 文件下的名字 这个项目是 helloworld 之后
helloworld_node 是 一个这个项目的映射名称 就算我们的 c++文件名字是什么并不关键但我们要有东西 136 add_executable 的第一个参数 ${PROJECT_NAME}_node
这个地方发生变化 需要在 149 也一起跟这·变化

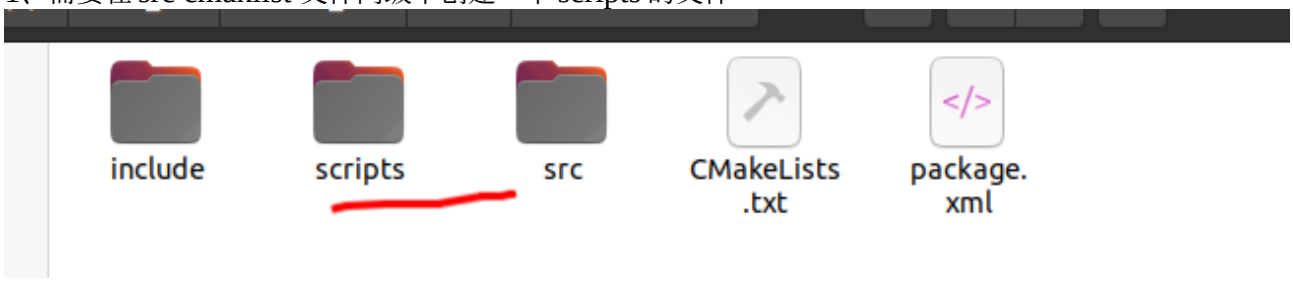


之后 info 这就算 内置的为在

输出日志

ros 中执行 python 文件

1、需要在 src cmaklist 文件同级下创建一个 scripts 的文件

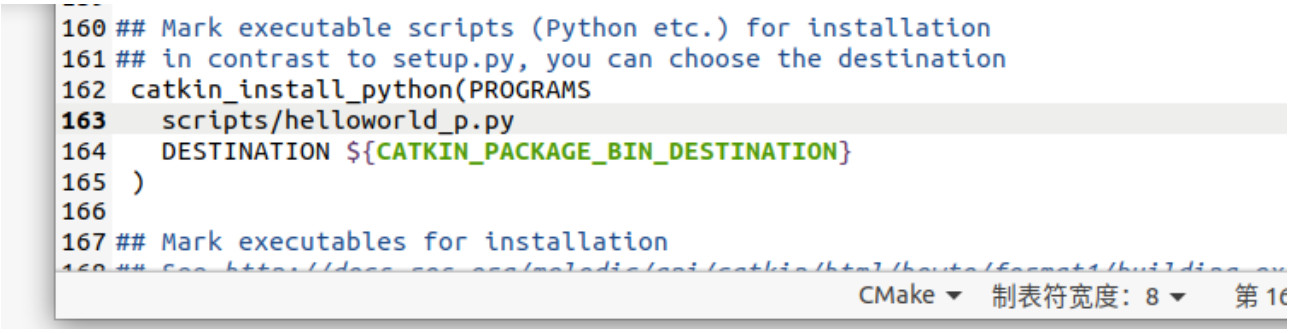


2、在 scripts 文件夹下面写一个 py 文件

```python
#指定解释器
#bin env python

import rospy
#导入ros包


#编写main函数 主函数
if __name__ == "__main__":
        # 初始化ros的节点
        rospy.init_node("hello")
        # 输出日志
        rospy.loginfo("hello world python")
```

要在 scripts 文件夹下面打开终端,
chmod +x helloworld_p.py 给他执行权限

162 行 告诉 ros 我们要执行的 python 文件是哪一个



之后就算在 demo_01 的路径下
caktin_make
新打开一个客户端 rocore 运行 ros

在 demo_01 的路径打开的终端下
source ./devel/setup.bash

rosrun helloworld helloworld_p.oy 运行 python 文件
rosrun ros 运行 关键名
helloworld 包名
helloworld_p.py python 文件名字