1. **GITHUB**

**Share repository**

1. Create repository on Github
2. Setting
3. Manage Access
4. Invite a collaborator
5. Write email and send

**Download Repository**

**open the terminal**

1. **cd workspace/src**
2. **git clone link the repository**

**Upload Repository for the first time**

**Open terminal, go to the folder that we want upload in this case the folder of name test**

1. **cd workspace/src /test**
2. git init
3. git add .
4. git config user.email "[nabihandres@gmail.com](mailto:nabihandres@gmail.com)"
5. git commit -m "first commit"
6. git remote add origin <https://github.com/XXXXXXX.git>
7. git push -u origin master
8. username
9. password

Note: if create submodules in the repository it mean the repository previous you make git init and have the folder inside, we must delete this folder with ctrl+h in the folder to upload and delete the folder git init and repeat the steps

**Upload Repository for Second or more time**

**Open terminal, go to the folder that we want upload for second time in this case the folder of name test**

1. cd workspace/scr /test
2. git add .
3. git config user.email "[xxxx@gmail.com](mailto:xxxx@gmail.com)"
4. git commit -m "second commit"
5. git push -u origin master
6. username
7. password

**Updated the repository**

**Note:** when is shared repo,If the github repo is pushed with new changes and your local repo doesn't have that changes this error occurs for this you could use:

1. git pull --rebase origin master

2 ) git push origin master

1. **C++**

**Run code C++ in ROS**

**Create a workspace**

1. mkdir -p ~/catkin\_ws/src
2. cd ~/catkin\_ws/
3. catkin\_make

**Create a package**

1. catkin\_ws/src
2. catkin\_create\_pkg test std\_msgs rospy roscpp tf
3. catkin\_make

Note: Inside the folder test now are two folders include, src and **CMakeLists.txt** and **package.xml** files

**Code C++ in this case** talker

Dowland code in in Github

1. git clone <https://github.com/emontero1991/tutorial-c-.git>

**Modify the Cmakelist of the folder where is the code in cpp**

1)

catkin\_package(

INCLUDE\_DIRS include

LIBRARIES test

)

2)

add\_executable(talker src/ talker.cpp)

3)

add\_dependencies(talker ${${PROJECT\_NAME}\_EXPORTED\_TARGETS} ${catkin\_EXPORTED\_TARGETS})

4)

target\_link\_libraries(talker

${catkin\_LIBRARIES}

)

**Open new terminal and run roscore**

1. roscore

**Open second terminal**

1. catkin\_make
2. source devel/setup.bash
3. rosrun test talker

**Use Eigen Library in ROS**

**tutorial**

<https://www.youtube.com/watch?v=jna5J3HsJI0>

https://blog.naver.com/tmdals727/221374583966

**Modify the Cmakelist of the folder where is the code in cpp**

1)

## eigen

find\_package(cmake\_modules REQUIRED)

list(APPEND CMAKE\_MODULE\_PATH ${CMAKE\_CURRENT\_SOURCE\_DIR}/cmake)

find\_package(Eigen REQUIRED)

include\_directories(${Eigen\_INCLUDE\_DIRS})

# eigen

2)

catkin\_package(

INCLUDE\_DIRS include

LIBRARIES test

DEPENDS Eigen roscpp rospy tf # eigen

)

3)

add\_executable(eigen src/eigen.cpp)

4)

add\_dependencies(eigen ${${PROJECT\_NAME}\_EXPORTED\_TARGETS} ${catkin\_EXPORTED\_TARGETS})

5)

target\_link\_libraries(eigen

${catkin\_LIBRARIES}

${Eigen\_LIBRARIES} # eigen

)

1. **Launch Files**

Launch files are very common in ROS to both users and developers. They provide a convenient way to start up multiple nodes and a master, as well as other initialization requirements such as setting parameters.

**Create the folder launch inside your package**

1. cd test
2. mkdir launch

NOTE: The directory to store launch files doesn't necessarily have to be named launch. In fact you don't even need to store them in a directory. roslaunch command automatically looks into the passed package and detects available launch files. However, this is considered good practice.

**Write the launch file in sublime text**

Launch files are of the format .launch and use a specific XML format.

The contents of a launch file must be contained between a pair of launch tags

1. <launch> … </launch>

To actually start a node, the <node> tags are used, the pkg, type and name argument are required.

1. <node name=”…” pkg=”…” type=”…” respawn=true ns=”…”/>

name: write a name itself is a unique identifier

pkg: The name of the **package** associated with the node that is to be launch in this case test is the name of the pkg

type: refers to the name of the node executable file in this case talker because the file is talker.cpp

**<?xml version= "1.0" encoding= "UTF-8"?>**

**<launch>**

**<!-- helloworld node file name:talker.cpp-->**

**<arg name="node\_start\_delay" default="0.0" />**

**<node name="example" pkg="test" type="talker" respawn="false" output="screen" launch-prefix="bash -c 'sleep $(arg node\_start\_delay); $0 $@' "/>**

**</launch>**

1. Save the file with extension .launch in this case talker.launch in the folder launch

**Run Roslaunch in one terminal**

Write roslaunch tool for launching noder, then the name of the package, finally the launch file.

1. roslaunch test talker.launch