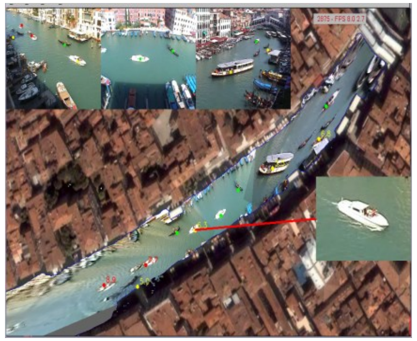
UNIVERSITÀ di VERONA

Dipartimento di INFORMATICA

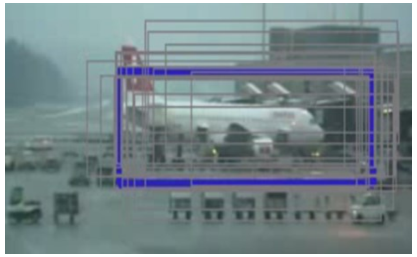
Corso di Laboratorio Ciberfisico
Modulo di Robot Programming with ROS

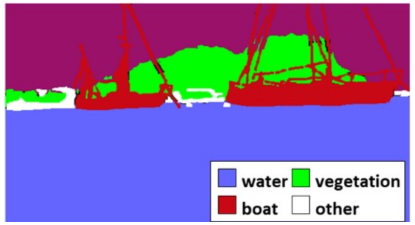


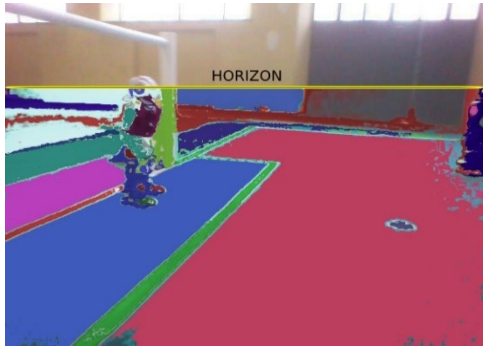
git + ROS

Docente:

Domenico Daniele Bloisi







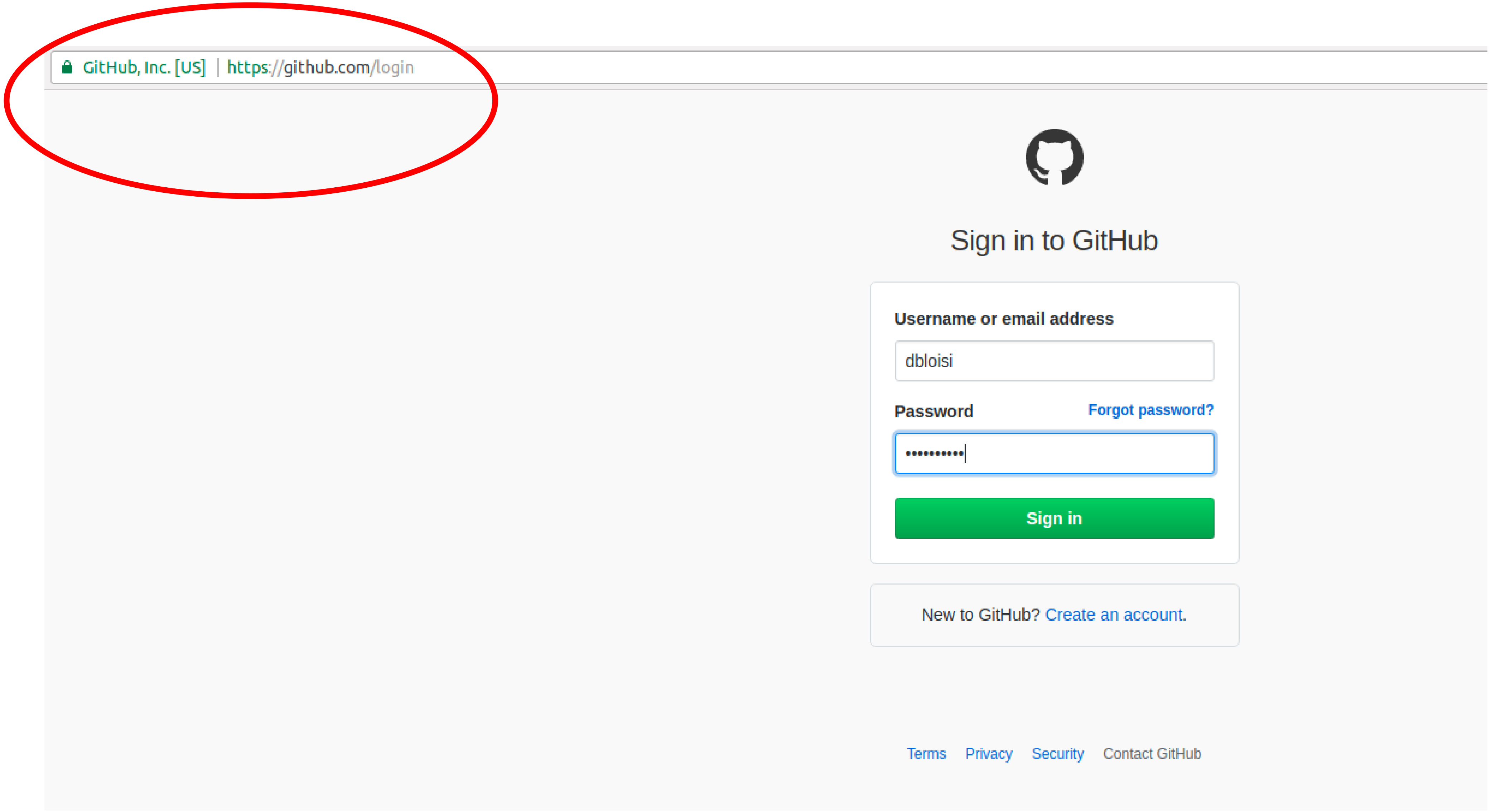
Marzo 2018

git + ROS

Esempio pratico

1. creare un repository git
2. creare un nodo ROS
3. condividere il nodo ROS tramite il repository git
4. modificare il nodo ROS usando git

Server git



GitHub, Inc. [US] | https://github.com/login

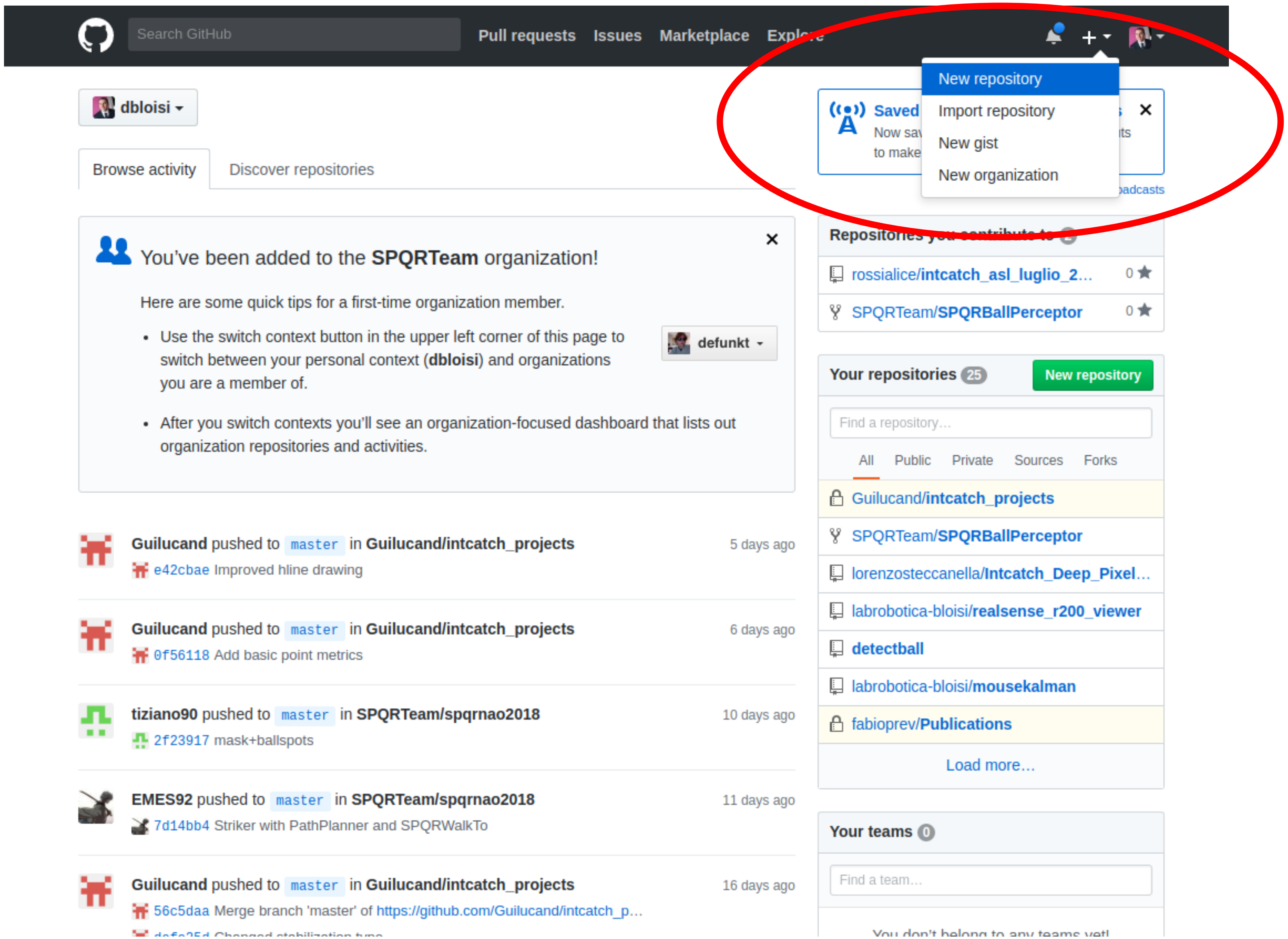
Sign in to GitHub

Username or email address: dbloisi

Password: ••••••••••

New to GitHub? Create an account.

Creare un repository git



Search GitHub Pull requests Issues Marketplace Explore

New repository
Import repository
New gist
New organization

dbloisi

Browse activity | Discover repositories

You've been added to the **SPQRTeam** organization!

Here are some quick tips for a first-time organization member.

- Use the switch context button in the upper left corner of this page to switch between your personal context (**dbloisi**) and organizations you are a member of.
- After you switch contexts you'll see an organization-focused dashboard that lists out organization repositories and activities.

defunkt

Guilucand pushed to `master` in **Guilucand/intcatch_projects** — 5 days ago
`e42cbae` Improved hline drawing

Guilucand pushed to `master` in **Guilucand/intcatch_projects** — 6 days ago
`0f56118` Add basic point metrics

tiziano90 pushed to `master` in **SPQRTeam/spqrnao2018** — 10 days ago
`2f23917` mask+ballspots

EMES92 pushed to `master` in **SPQRTeam/spqrnao2018** — 11 days ago
`7d14bb4` Striker with PathPlanner and SPQRWalkTo

Guilucand pushed to `master` in **Guilucand/intcatch_projects** — 16 days ago
`56c5daa` Merge branch 'master' of https://github.com/Guilucand/intcatch_p…
`dafe25d` Changed stabilization type

Saved … Now sa… to make … broadcasts

Repositories you contribute to 2

- rossialice/**intcatch_asl_luglio_2…** 0 ★
- SPQRTeam/**SPQRBallPerceptor** 0 ★

Your repositories 25 — New repository

Find a repository…

All | Public | Private | Sources | Forks

- Guilucand/**intcatch_projects**
- SPQRTeam/**SPQRBallPerceptor**
- lorenzosteccanella/**Intcatch_Deep_Pixel…**
- labrobotica-bloisi/**realsense_r200_viewer**
- **detectball**
- labrobotica-bloisi/**mousekalman**
- fabioprev/**Publications**

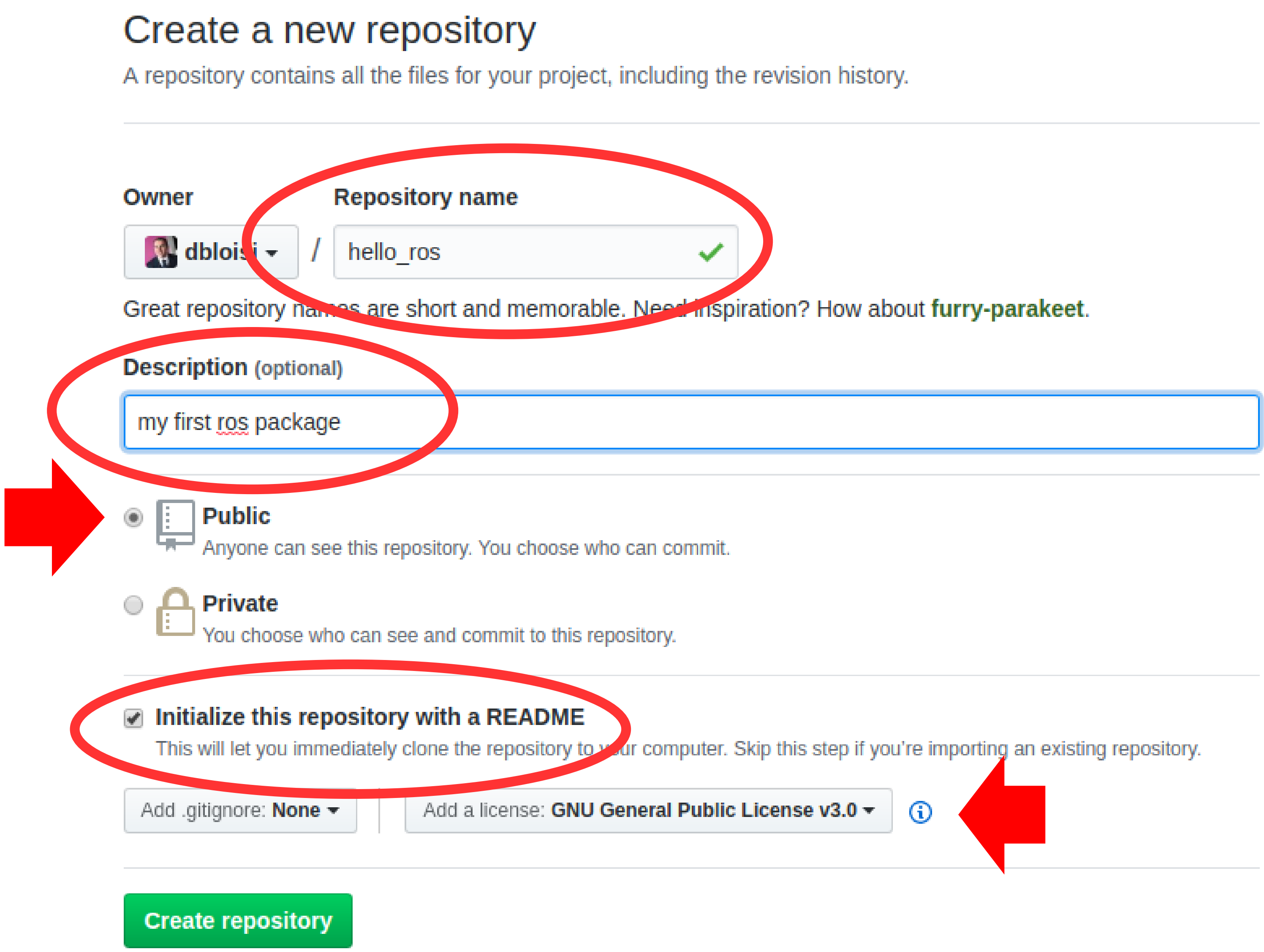
Load more…

Your teams 0

Find a team…

You don't belong to any teams yet!

Repository name



Create a new repository

A repository contains all the files for your project, including the revision history.

Owner dbloisi / **Repository name** hello_ros ✓

Great repository names are short and memorable. Need inspiration? How about **furry-parakeet**.

Description (optional)

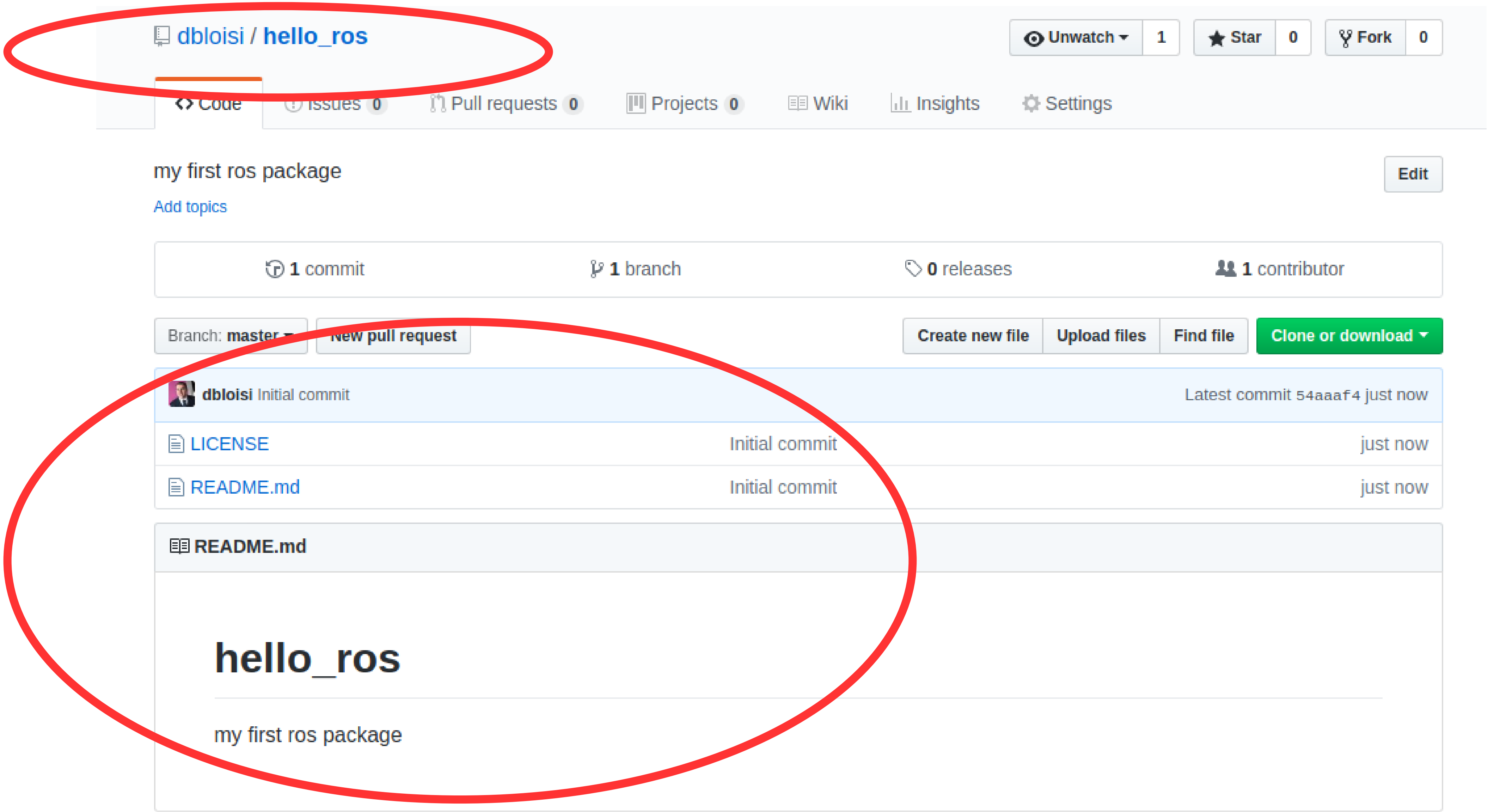
my first ros package

● **Public**
Anyone can see this repository. You choose who can commit.

○ **Private**
You choose who can see and commit to this repository.

☑ **Initialize this repository with a README**
This will let you immediately clone the repository to your computer. Skip this step if you're importing an existing repository.

Repository creato



my first ros package

dbloisi Initial commit — Latest commit 54aaaf4 just now

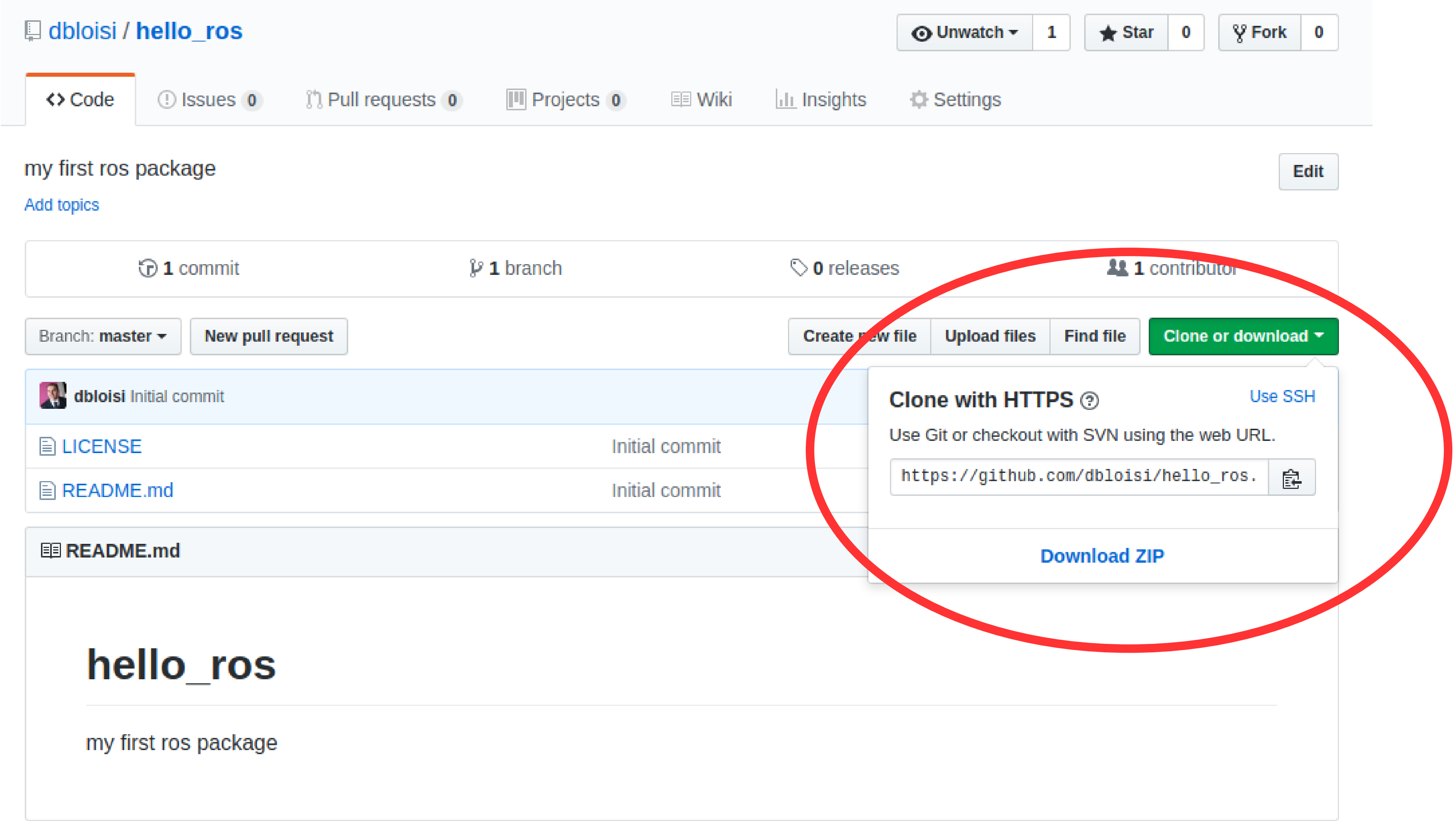
LICENSE	Initial commit	just now
README.md	Initial commit	just now

README.md

hello_ros

my first ros package

Indirizzo del repository remoto



my first ros package

dbloisi Initial commit

LICENSE	Initial commit
README.md	Initial commit

README.md

hello_ros

my first ros package

Clone with HTTPS

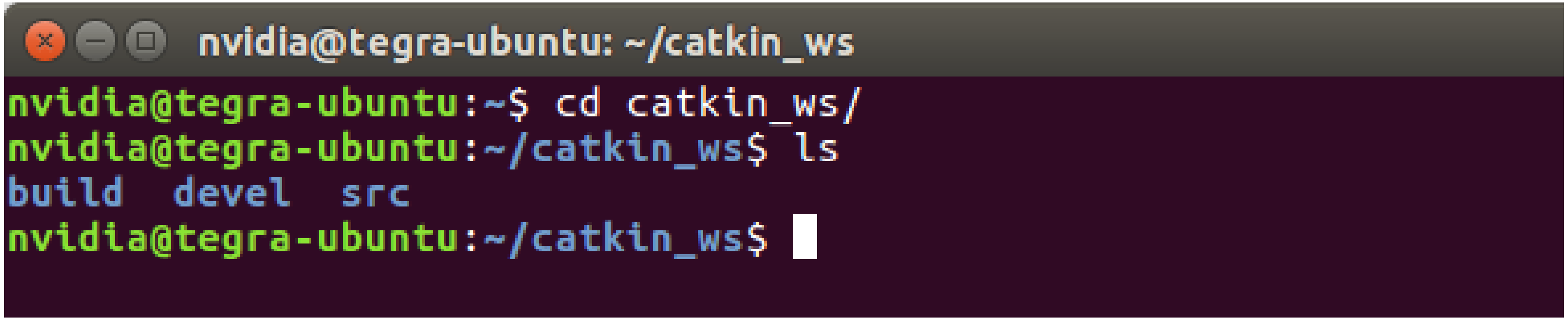
Use Git or checkout with SVN using the web URL.

```
https://github.com/dbloisi/hello_ros.
```

Creazione del repository locale

Il repository remoto si trova in https://github.com/dbloisi/hello_ros

Creiamo il repository locale nel nostro workspace ROS ~/catkin_ws

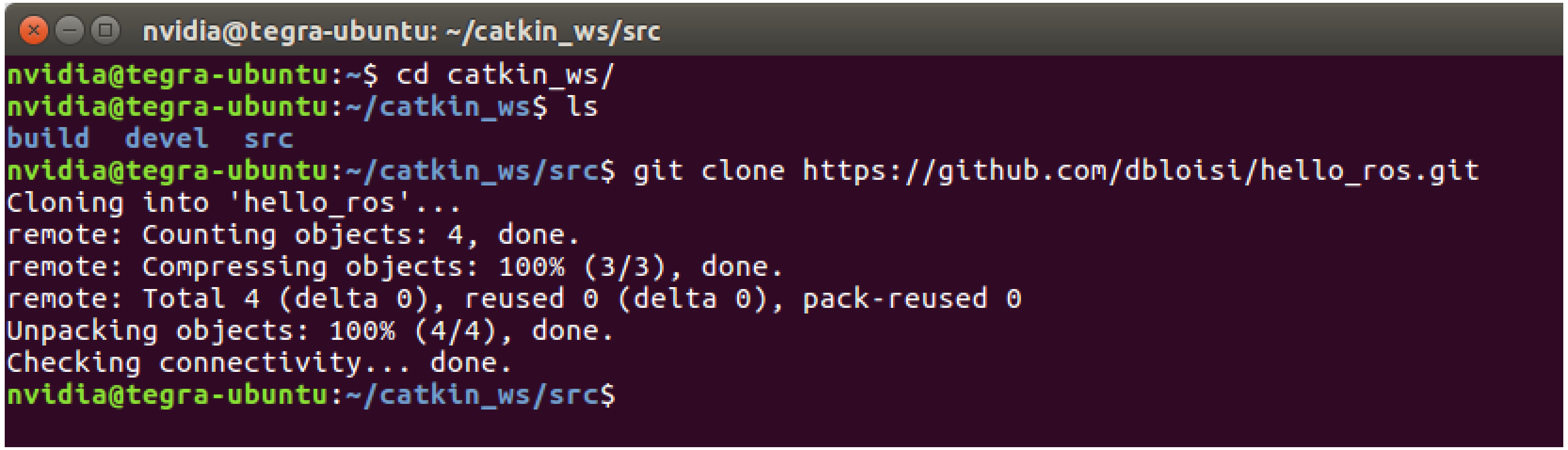


```
nvidia@tegra-ubuntu:~$ cd catkin_ws/
nvidia@tegra-ubuntu:~/catkin_ws$ ls
build  devel  src
nvidia@tegra-ubuntu:~/catkin_ws$ 
```

Creazione del repository locale

Il repository remoto si trova in https://github.com/dbloisi/hello_ros

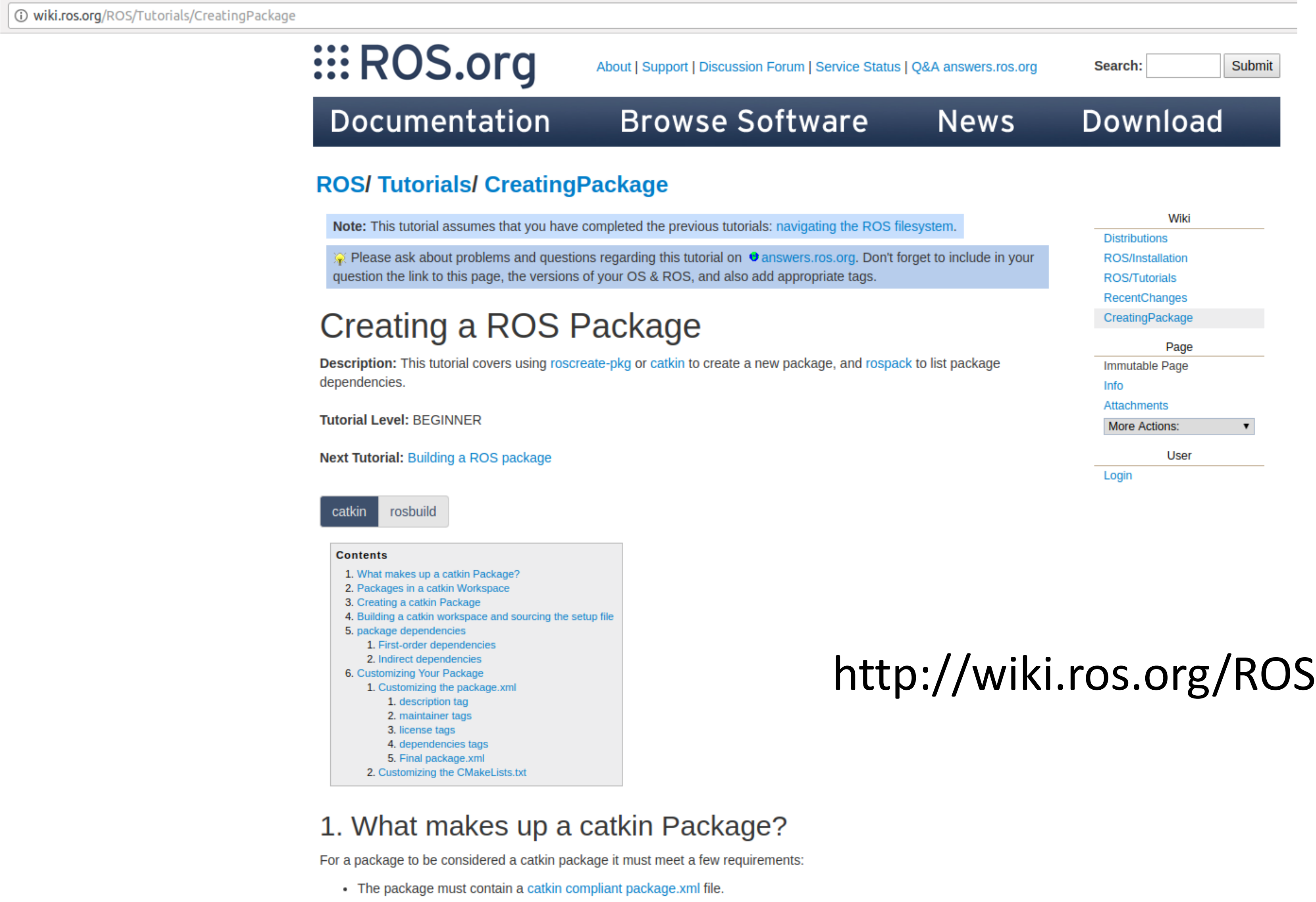
Il repository locale sarà creato in ~/catkin_ws/src/hello_ros



nvidia@tegra-ubuntu: ~/catkin_ws/src

```
nvidia@tegra-ubuntu:~$ cd catkin_ws/
nvidia@tegra-ubuntu:~/catkin_ws$ ls
build  devel  src
nvidia@tegra-ubuntu:~/catkin_ws/src$ git clone https://github.com/dbloisi/hello_ros.git
Cloning into 'hello_ros'...
remote: Counting objects: 4, done.
remote: Compressing objects: 100% (3/3), done.
remote: Total 4 (delta 0), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (4/4), done.
Checking connectivity... done.
nvidia@tegra-ubuntu:~/catkin_ws/src$
```

Creating a ROS package



ROS/ Tutorials/ CreatingPackage

Note: This tutorial assumes that you have completed the previous tutorials: navigating the ROS filesystem.

Please ask about problems and questions regarding this tutorial on answers.ros.org. Don't forget to include in your question the link to this page, the versions of your OS & ROS, and also add appropriate tags.

Creating a ROS Package

Description: This tutorial covers using roscreate-pkg or catkin to create a new package, and rospack to list package dependencies.

Tutorial Level: BEGINNER

Next Tutorial: Building a ROS package

Contents

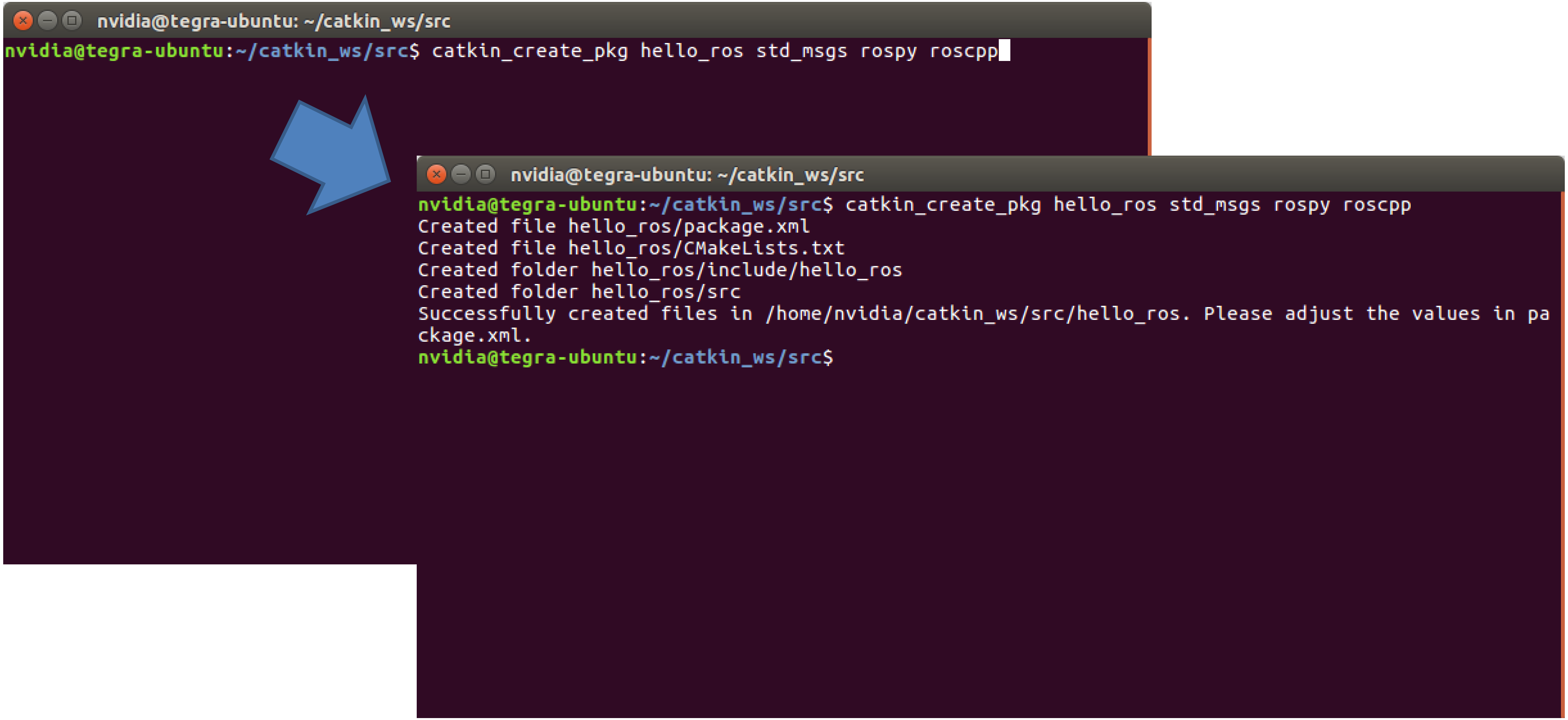
1. What makes up a catkin Package?

For a package to be considered a catkin package it must meet a few requirements:

- The package must contain a catkin compliant package.xml file.

http://wiki.ros.org/ROS/Tutorials/CreatingPackage

catkin_create_pkg



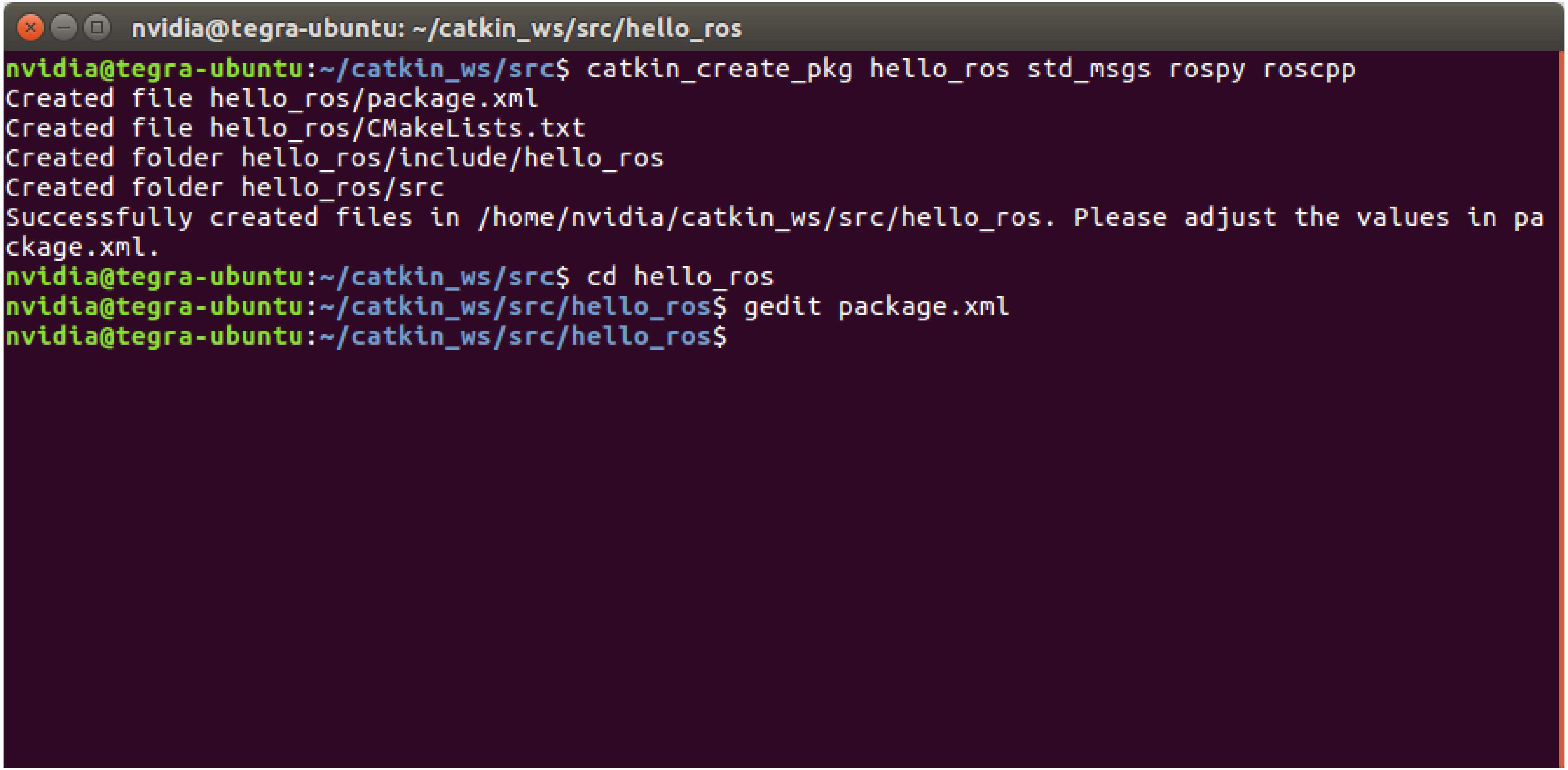
nvidia@tegra-ubuntu: ~/catkin_ws/src

```
nvidia@tegra-ubuntu:~/catkin_ws/src$ catkin_create_pkg hello_ros std_msgs rospy roscpp
```

nvidia@tegra-ubuntu: ~/catkin_ws/src

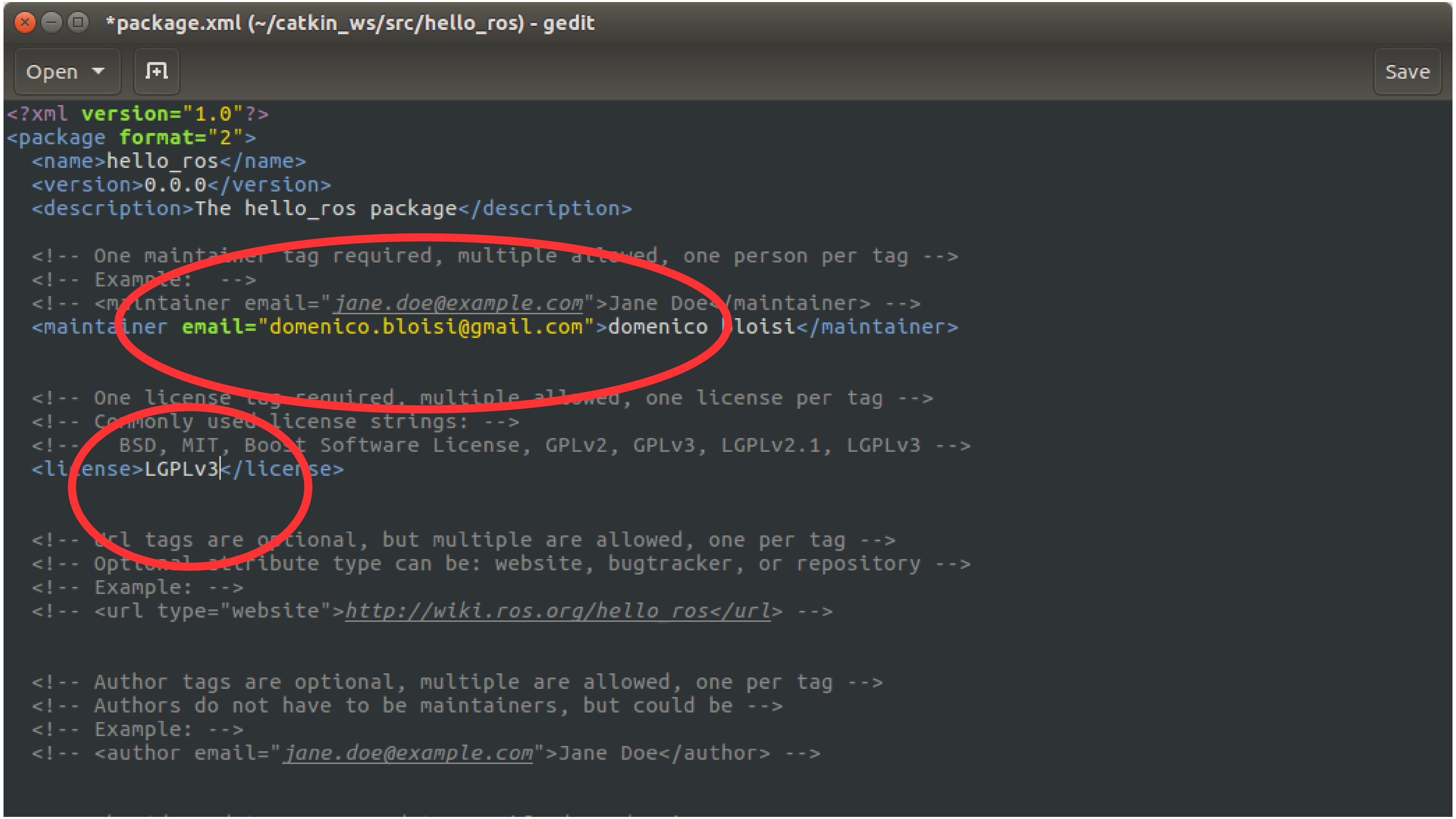
```
nvidia@tegra-ubuntu:~/catkin_ws/src$ catkin_create_pkg hello_ros std_msgs rospy roscpp
Created file hello_ros/package.xml
Created file hello_ros/CMakeLists.txt
Created folder hello_ros/include/hello_ros
Created folder hello_ros/src
Successfully created files in /home/nvidia/catkin_ws/src/hello_ros. Please adjust the values in pa
ckage.xml.
nvidia@tegra-ubuntu:~/catkin_ws/src$
```

package.xml



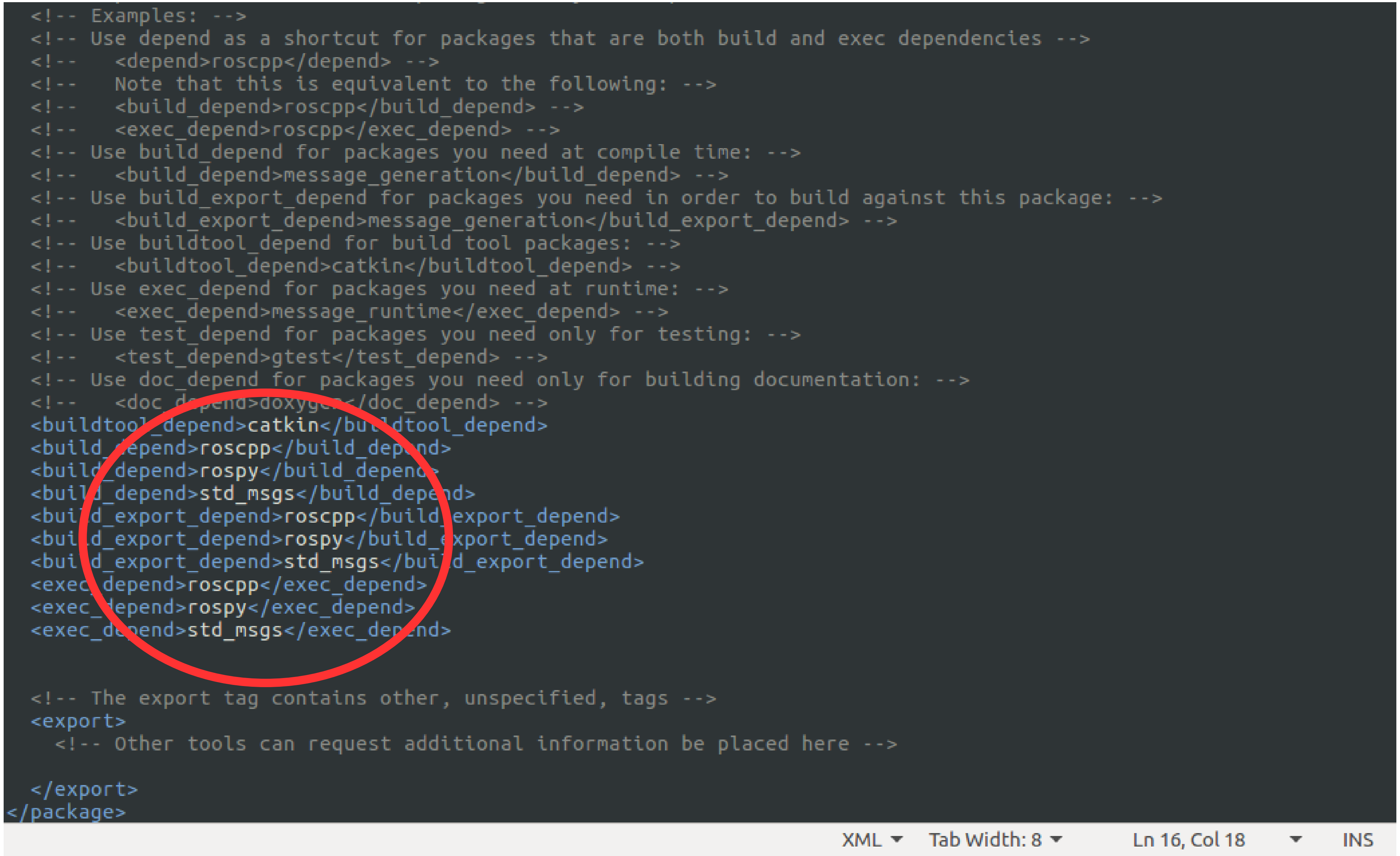
```
nvidia@tegra-ubuntu:~/catkin_ws/src$ catkin_create_pkg hello_ros std_msgs rospy roscpp
Created file hello_ros/package.xml
Created file hello_ros/CMakeLists.txt
Created folder hello_ros/include/hello_ros
Created folder hello_ros/src
Successfully created files in /home/nvidia/catkin_ws/src/hello_ros. Please adjust the values in pa
ckage.xml.
nvidia@tegra-ubuntu:~/catkin_ws/src$ cd hello_ros
nvidia@tegra-ubuntu:~/catkin_ws/src/hello_ros$ gedit package.xml
nvidia@tegra-ubuntu:~/catkin_ws/src/hello_ros$
```

Inserimento dati in package.xml



```
<?xml version="1.0"?>
<package format="2">
  <name>hello_ros</name>
  <version>0.0.0</version>
  <description>The hello_ros package</description>

  <!-- One maintainer tag required, multiple allowed, one person per tag -->
  <!-- Example:  -->
  <!-- <maintainer email="jane.doe@example.com">Jane Doe</maintainer> -->
  <maintainer email="domenico.bloisi@gmail.com">domenico bloisi</maintainer>


  <!-- One license tag required, multiple allowed, one license per tag -->
  <!-- Commonly used license strings: -->
  <!--   BSD, MIT, Boost Software License, GPLv2, GPLv3, LGPLv2.1, LGPLv3 -->
  <license>LGPLv3</license>


  <!-- Url tags are optional, but multiple are allowed, one per tag -->
  <!-- Optional attribute type can be: website, bugtracker, or repository -->
  <!-- Example: -->
  <!-- <url type="website">http://wiki.ros.org/hello_ros</url> -->


  <!-- Author tags are optional, multiple are allowed, one per tag -->
  <!-- Authors do not have to be maintainers, but could be -->
  <!-- Example: -->
  <!-- <author email="jane.doe@example.com">Jane Doe</author> -->
```

Dipendenze in package.xml

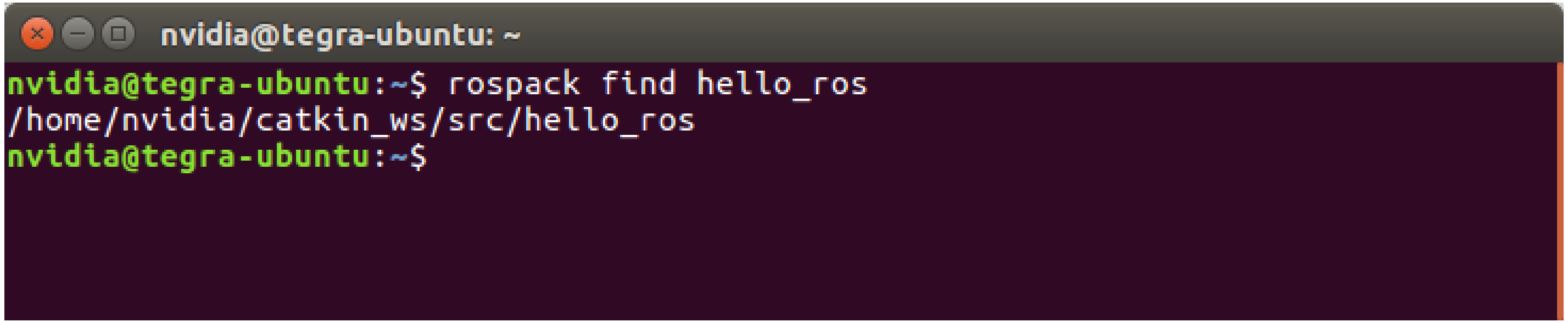


```
  <!-- Examples: -->
  <!-- Use depend as a shortcut for packages that are both build and exec dependencies -->
  <!--   <depend>roscpp</depend> -->
  <!--   Note that this is equivalent to the following: -->
  <!--   <build_depend>roscpp</build_depend> -->
  <!--   <exec_depend>roscpp</exec_depend> -->
  <!-- Use build_depend for packages you need at compile time: -->
  <!--   <build_depend>message_generation</build_depend> -->
  <!-- Use build_export_depend for packages you need in order to build against this package: -->
  <!--   <build_export_depend>message_generation</build_export_depend> -->
  <!-- Use buildtool_depend for build tool packages: -->
  <!--   <buildtool_depend>catkin</buildtool_depend> -->
  <!-- Use exec_depend for packages you need at runtime: -->
  <!--   <exec_depend>message_runtime</exec_depend> -->
  <!-- Use test_depend for packages you need only for testing: -->
  <!--   <test_depend>gtest</test_depend> -->
  <!-- Use doc_depend for packages you need only for building documentation: -->
  <!--   <doc_depend>doxygen</doc_depend> -->
  <buildtool_depend>catkin</buildtool_depend>
  <build_depend>roscpp</build_depend>
  <build_depend>rospy</build_depend>
  <build_depend>std_msgs</build_depend>
  <build_export_depend>roscpp</build_export_depend>
  <build_export_depend>rospy</build_export_depend>
  <build_export_depend>std_msgs</build_export_depend>
  <exec_depend>roscpp</exec_depend>
  <exec_depend>rospy</exec_depend>
  <exec_depend>std_msgs</exec_depend>


  <!-- The export tag contains other, unspecified, tags -->
  <export>
    <!-- Other tools can request additional information be placed here -->

  </export>
</package>
```

Finding a ROS package

Now that your package has a manifest, ROS can find it. Try executing the command:

```
rospack find hello_ros
```

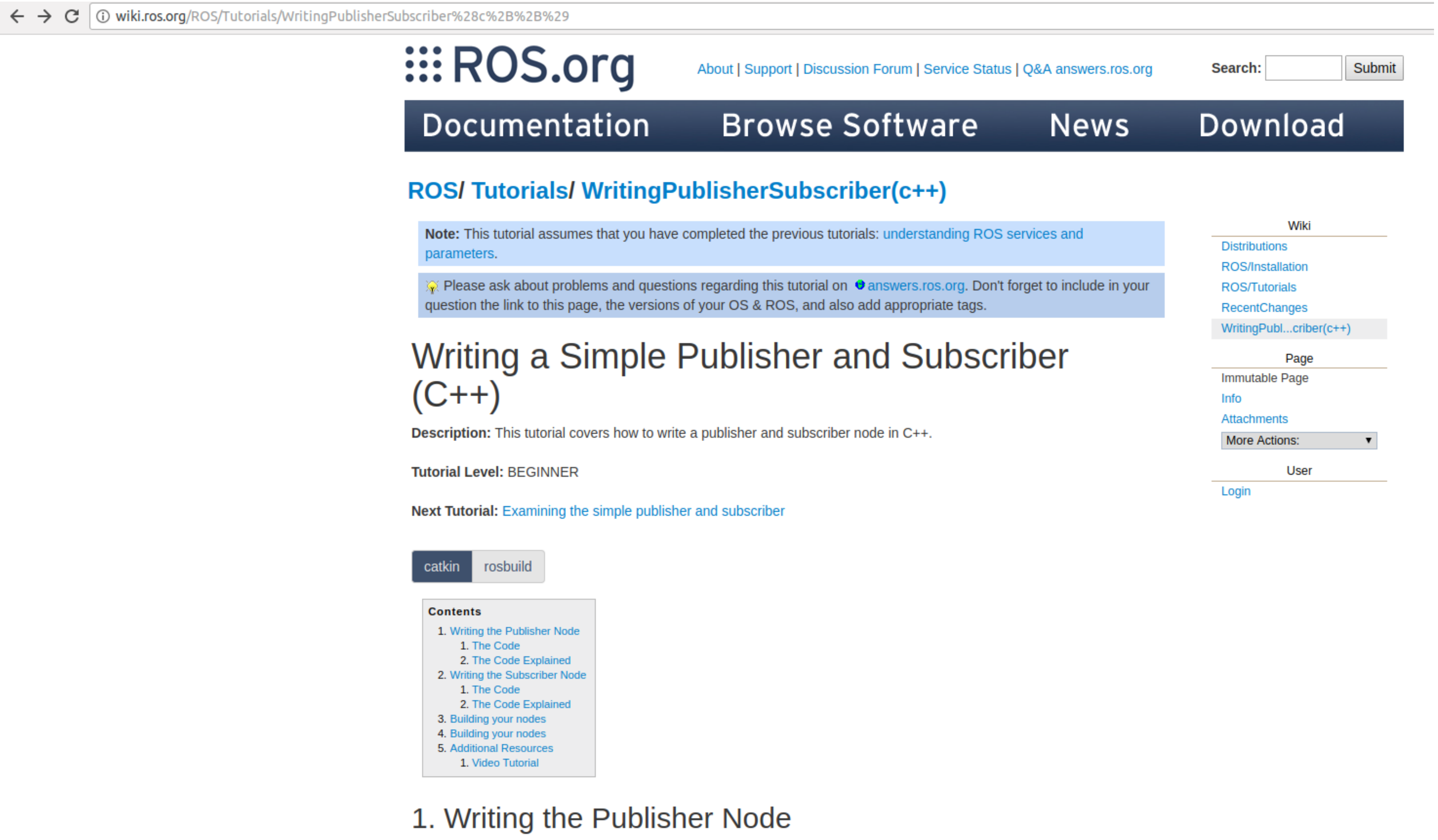


```
nvidia@tegra-ubuntu:~$ rospack find hello_ros
/home/nvidia/catkin_ws/src/hello_ros
nvidia@tegra-ubuntu:~$
```

if ROS is set up correctly you should see the physical location where your package is stored

http://wiki.ros.org/ROS/Tutorials/Creating%20a%20Package%20by%20Hand

Esempio Publisher/Subscriber C++



ROS/ Tutorials/ WritingPublisherSubscriber(c++)

Note: This tutorial assumes that you have completed the previous tutorials: understanding ROS services and parameters.

Please ask about problems and questions regarding this tutorial on answers.ros.org. Don't forget to include in your question the link to this page, the versions of your OS & ROS, and also add appropriate tags.

Writing a Simple Publisher and Subscriber (C++)

Description: This tutorial covers how to write a publisher and subscriber node in C++.

Tutorial Level: BEGINNER

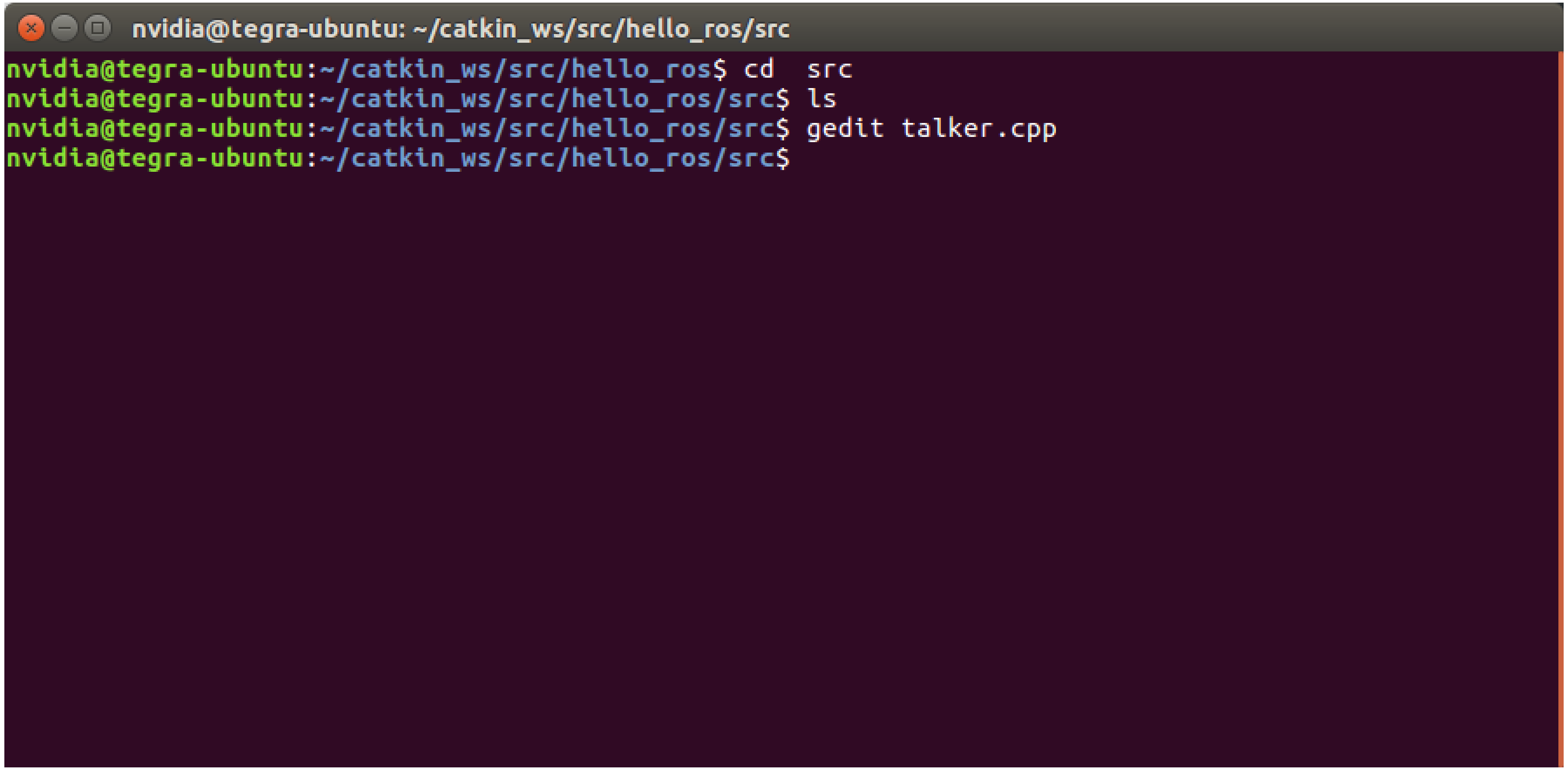
Next Tutorial: Examining the simple publisher and subscriber

Contents

1. Writing the Publisher Node

http://wiki.ros.org/ROS/Tutorials/WritingPublisherSubscriber%28c%2B%2B%29

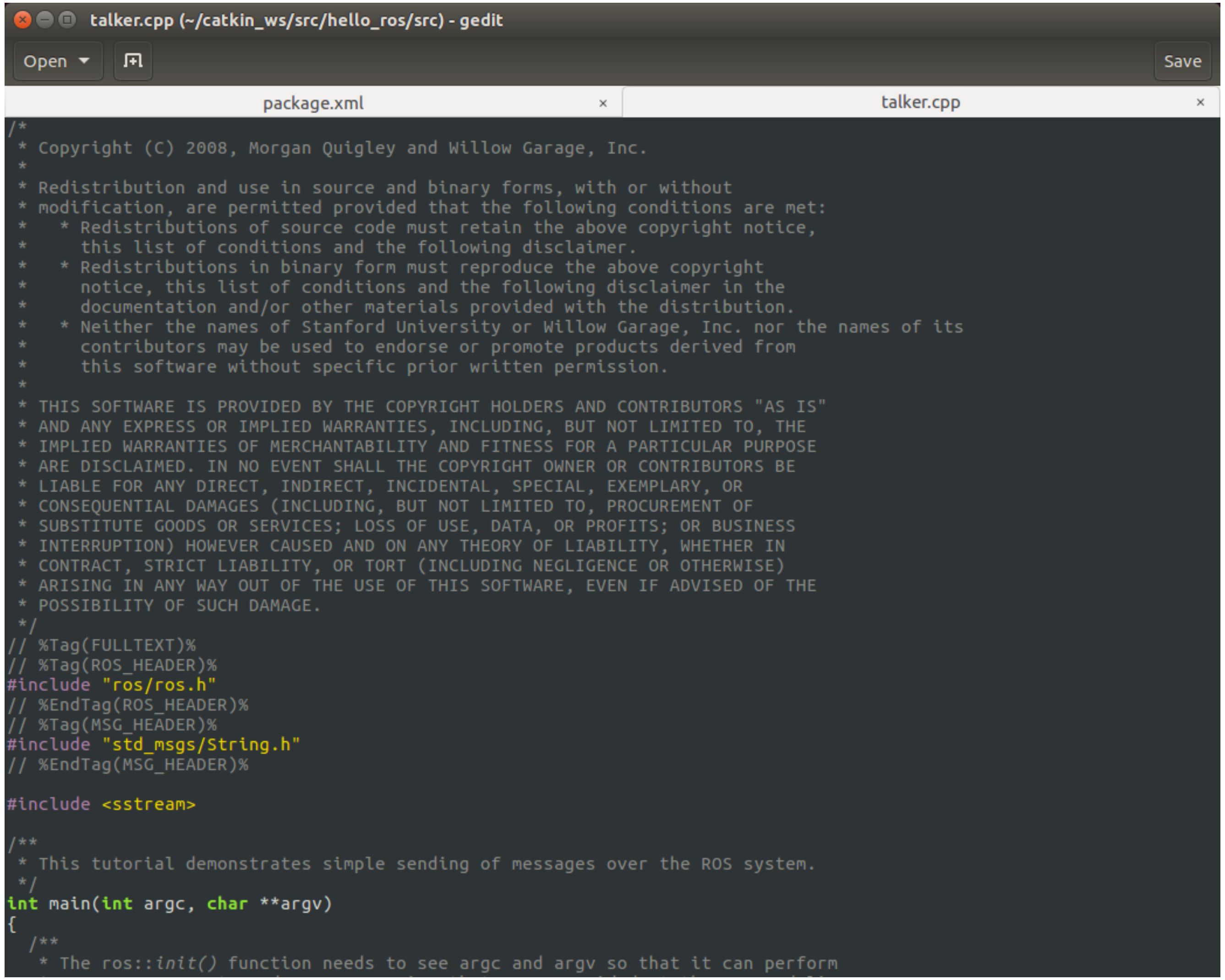
Creiamo il publisher (talker.cpp)



nvidia@tegra-ubuntu: ~/catkin_ws/src/hello_ros/src

```
nvidia@tegra-ubuntu:~/catkin_ws/src/hello_ros$ cd  src
nvidia@tegra-ubuntu:~/catkin_ws/src/hello_ros/src$ ls
nvidia@tegra-ubuntu:~/catkin_ws/src/hello_ros/src$ gedit talker.cpp
nvidia@tegra-ubuntu:~/catkin_ws/src/hello_ros/src$
```

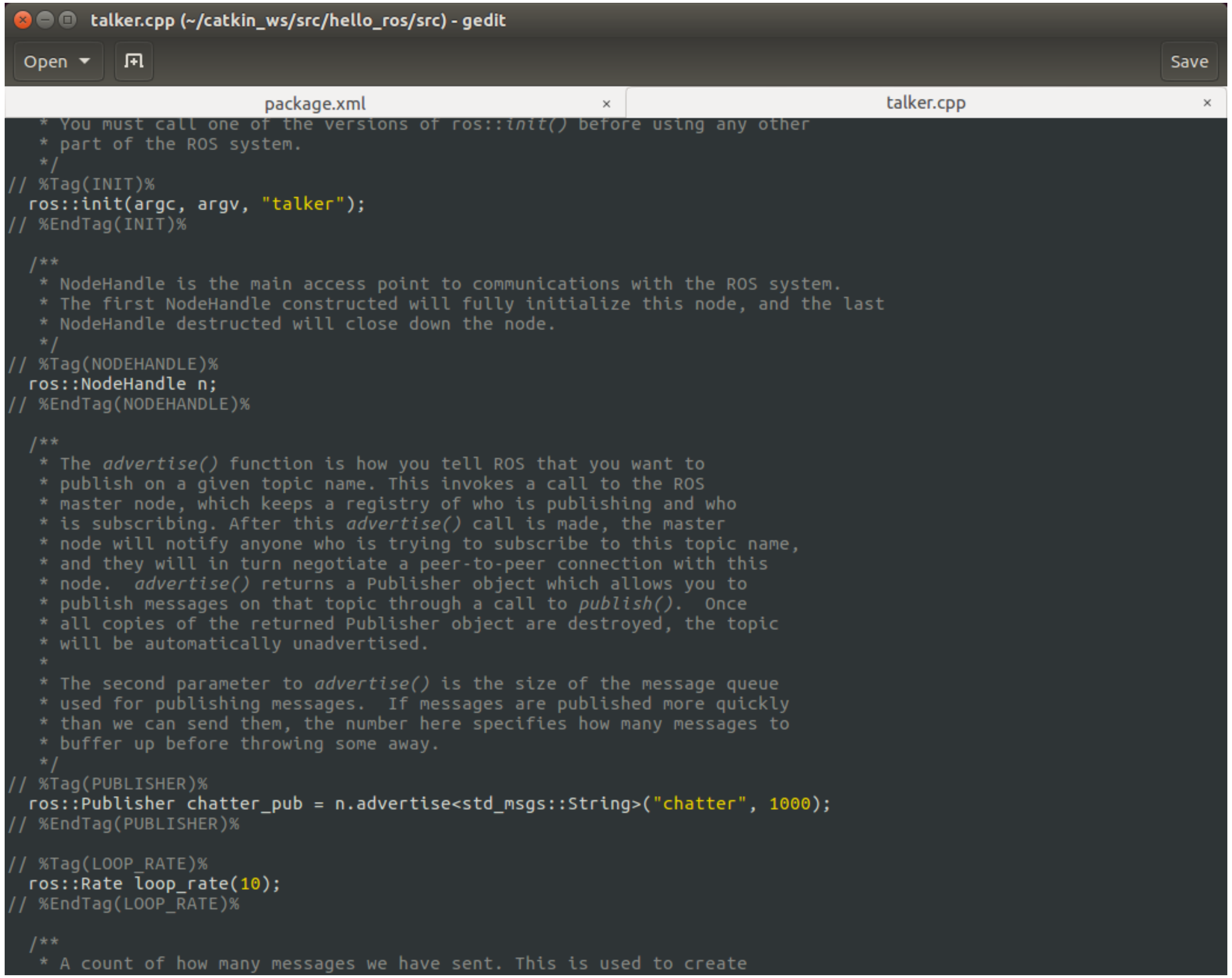
Codice del publisher (talker.cpp)



```cpp
/*
 * Copyright (C) 2008, Morgan Quigley and Willow Garage, Inc.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions are met:
 *   * Redistributions of source code must retain the above copyright notice,
 *     this list of conditions and the following disclaimer.
 *   * Redistributions in binary form must reproduce the above copyright
 *     notice, this list of conditions and the following disclaimer in the
 *     documentation and/or other materials provided with the distribution.
 *   * Neither the names of Stanford University or Willow Garage, Inc. nor the names of its
 *     contributors may be used to endorse or promote products derived from
 *     this software without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
 * AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
 * LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
 * CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
 * SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
 * INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
 * CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
 * ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
 * POSSIBILITY OF SUCH DAMAGE.
 */
// %Tag(FULLTEXT)%
// %Tag(ROS_HEADER)%
#include "ros/ros.h"
// %EndTag(ROS_HEADER)%
// %Tag(MSG_HEADER)%
#include "std_msgs/String.h"
// %EndTag(MSG_HEADER)%

#include <sstream>

/**
 * This tutorial demonstrates simple sending of messages over the ROS system.
 */
int main(int argc, char **argv)
{
  /**
   * The ros::init() function needs to see argc and argv so that it can perform
```

https://raw.githubusercontent.com/ros/ros_tutorials/kinetic-devel/roscpp_tutorials/talker/talker.cpp

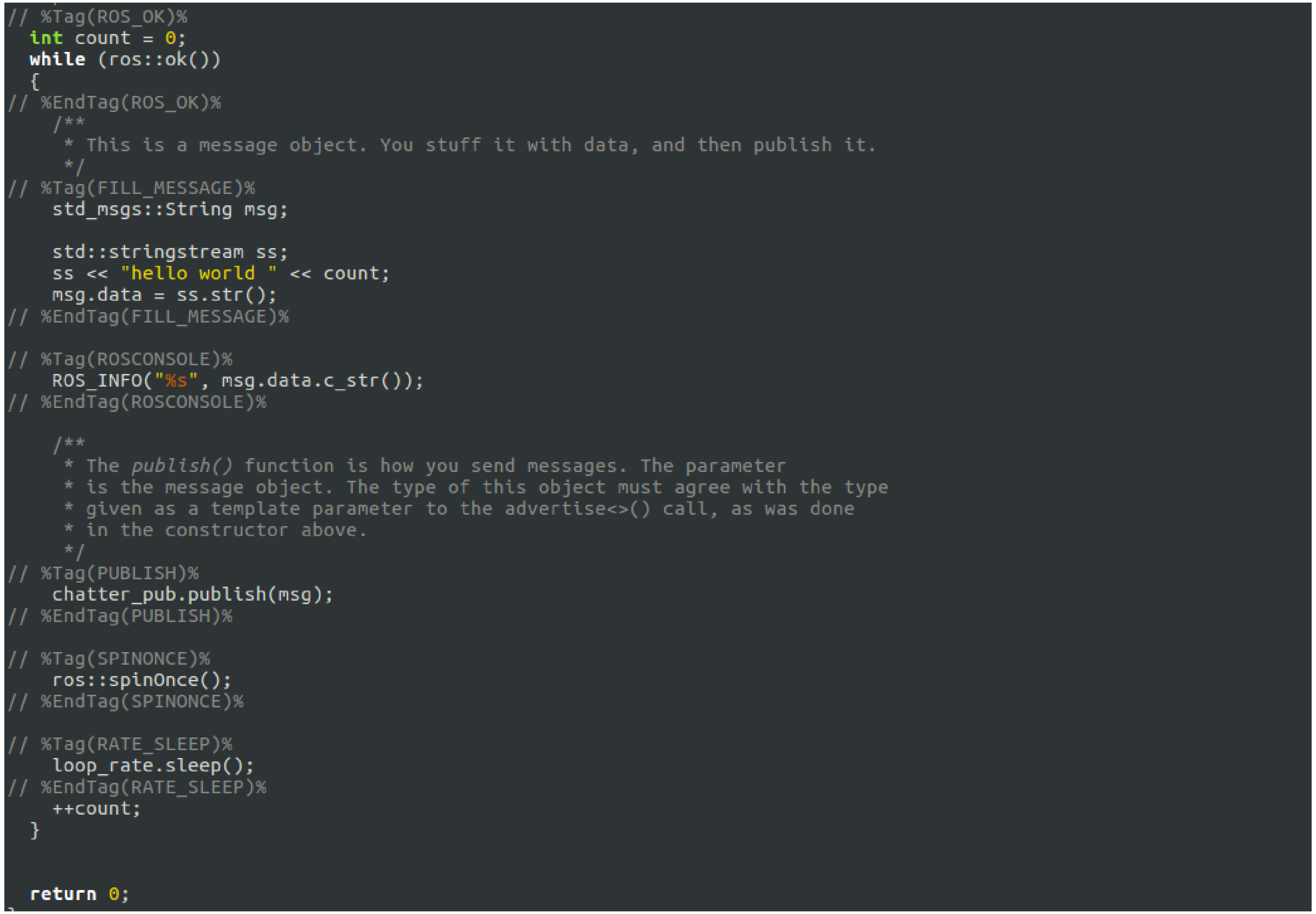
Codice del publisher (talker.cpp)



```
   * You must call one of the versions of ros::init() before using any other
   * part of the ROS system.
   */
// %Tag(INIT)%
  ros::init(argc, argv, "talker");
// %EndTag(INIT)%

  /**
   * NodeHandle is the main access point to communications with the ROS system.
   * The first NodeHandle constructed will fully initialize this node, and the last
   * NodeHandle destructed will close down the node.
   */
// %Tag(NODEHANDLE)%
  ros::NodeHandle n;
// %EndTag(NODEHANDLE)%

  /**
   * The advertise() function is how you tell ROS that you want to
   * publish on a given topic name. This invokes a call to the ROS
   * master node, which keeps a registry of who is publishing and who
   * is subscribing. After this advertise() call is made, the master
   * node will notify anyone who is trying to subscribe to this topic name,
   * and they will in turn negotiate a peer-to-peer connection with this
   * node.  advertise() returns a Publisher object which allows you to
   * publish messages on that topic through a call to publish().  Once
   * all copies of the returned Publisher object are destroyed, the topic
   * will be automatically unadvertised.
   *
   * The second parameter to advertise() is the size of the message queue
   * used for publishing messages.  If messages are published more quickly
   * than we can send them, the number here specifies how many messages to
   * buffer up before throwing some away.
   */
// %Tag(PUBLISHER)%
  ros::Publisher chatter_pub = n.advertise<std_msgs::String>("chatter", 1000);
// %EndTag(PUBLISHER)%

// %Tag(LOOP_RATE)%
  ros::Rate loop_rate(10);
// %EndTag(LOOP_RATE)%

  /**
   * A count of how many messages we have sent. This is used to create
```

https://raw.githubusercontent.com/ros/ros_tutorials/kinetic-devel/roscpp_tutorials/talker/talker.cpp

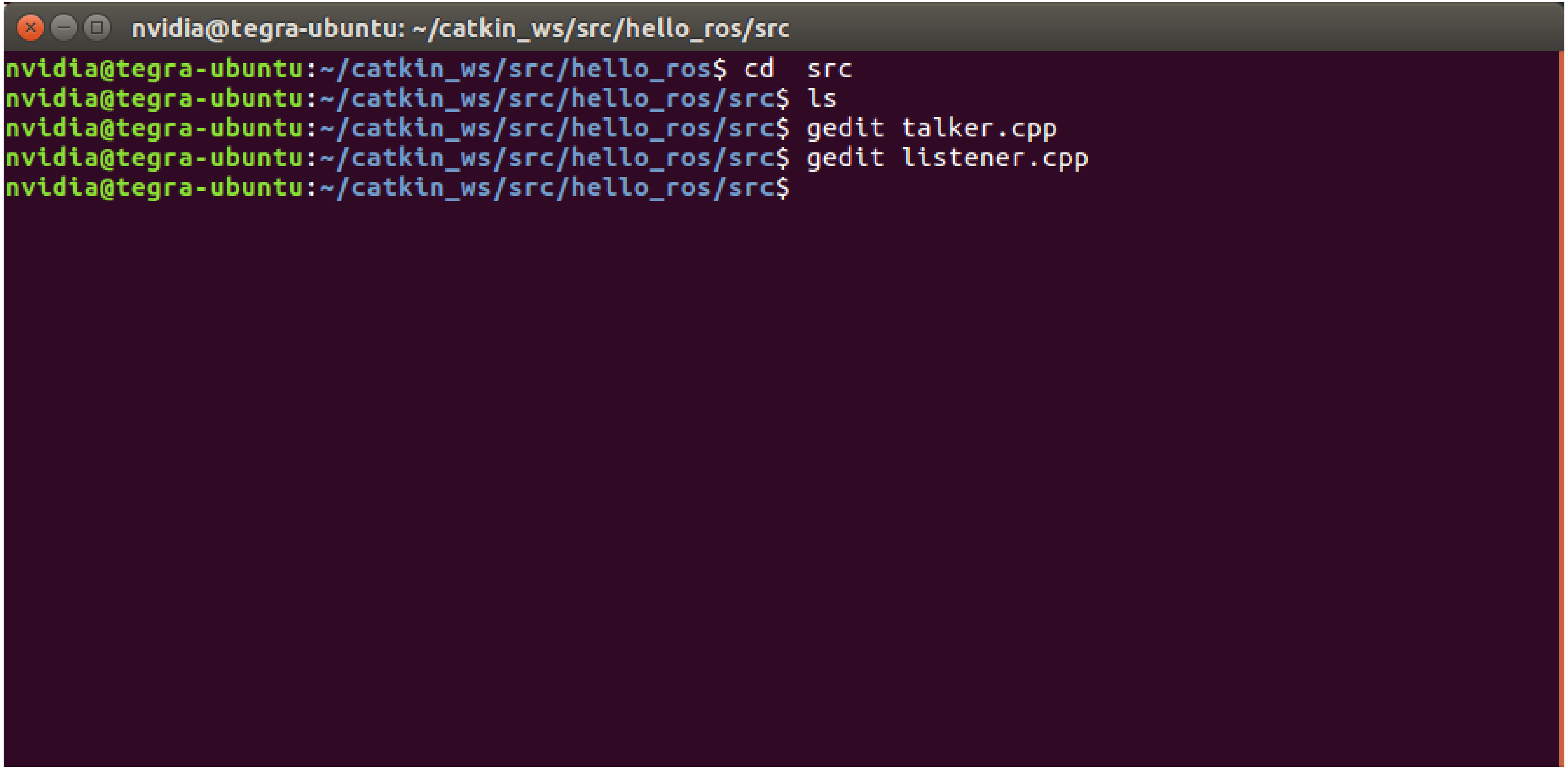
Codice del publisher (talker.cpp)



```
// %Tag(ROS_OK)%
  int count = 0;
  while (ros::ok())
  {
// %EndTag(ROS_OK)%
    /**
     * This is a message object. You stuff it with data, and then publish it.
     */
// %Tag(FILL_MESSAGE)%
    std_msgs::String msg;

    std::stringstream ss;
    ss << "hello world " << count;
    msg.data = ss.str();
// %EndTag(FILL_MESSAGE)%

// %Tag(ROSCONSOLE)%
    ROS_INFO("%s", msg.data.c_str());
// %EndTag(ROSCONSOLE)%

    /**
     * The publish() function is how you send messages. The parameter
     * is the message object. The type of this object must agree with the type
     * given as a template parameter to the advertise<>() call, as was done
     * in the constructor above.
     */
// %Tag(PUBLISH)%
    chatter_pub.publish(msg);
// %EndTag(PUBLISH)%

// %Tag(SPINONCE)%
    ros::spinOnce();
// %EndTag(SPINONCE)%

// %Tag(RATE_SLEEP)%
    loop_rate.sleep();
// %EndTag(RATE_SLEEP)%
    ++count;
  }


  return 0;
```

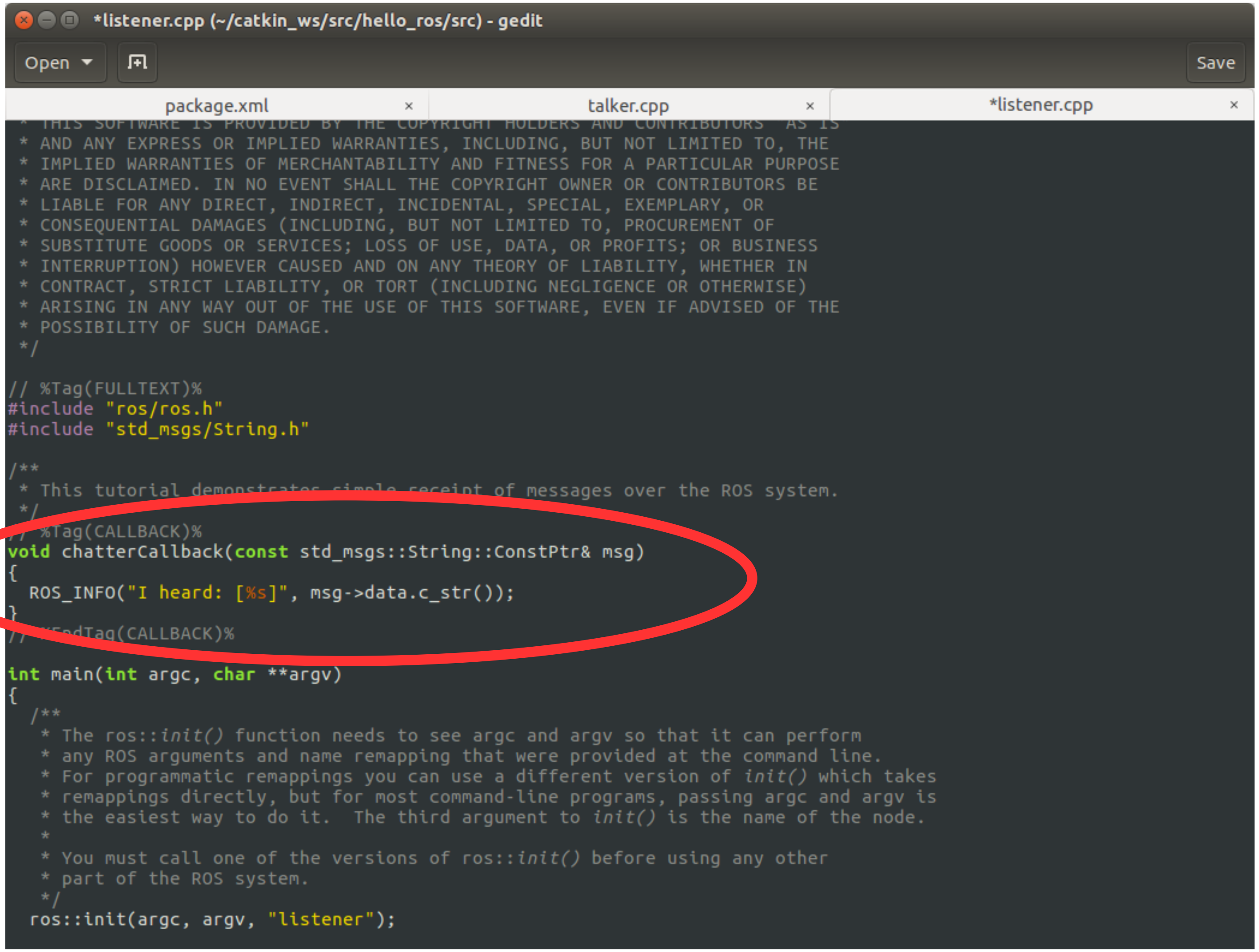
https://raw.githubusercontent.com/ros/ros_tutorials/kinetic-devel/roscpp_tutorials/talker/talker.cpp

Creiamo il subscriber (listener.cpp)



```
nvidia@tegra-ubuntu:~/catkin_ws/src/hello_ros$ cd  src
nvidia@tegra-ubuntu:~/catkin_ws/src/hello_ros/src$ ls
nvidia@tegra-ubuntu:~/catkin_ws/src/hello_ros/src$ gedit talker.cpp
nvidia@tegra-ubuntu:~/catkin_ws/src/hello_ros/src$ gedit listener.cpp
nvidia@tegra-ubuntu:~/catkin_ws/src/hello_ros/src$
```

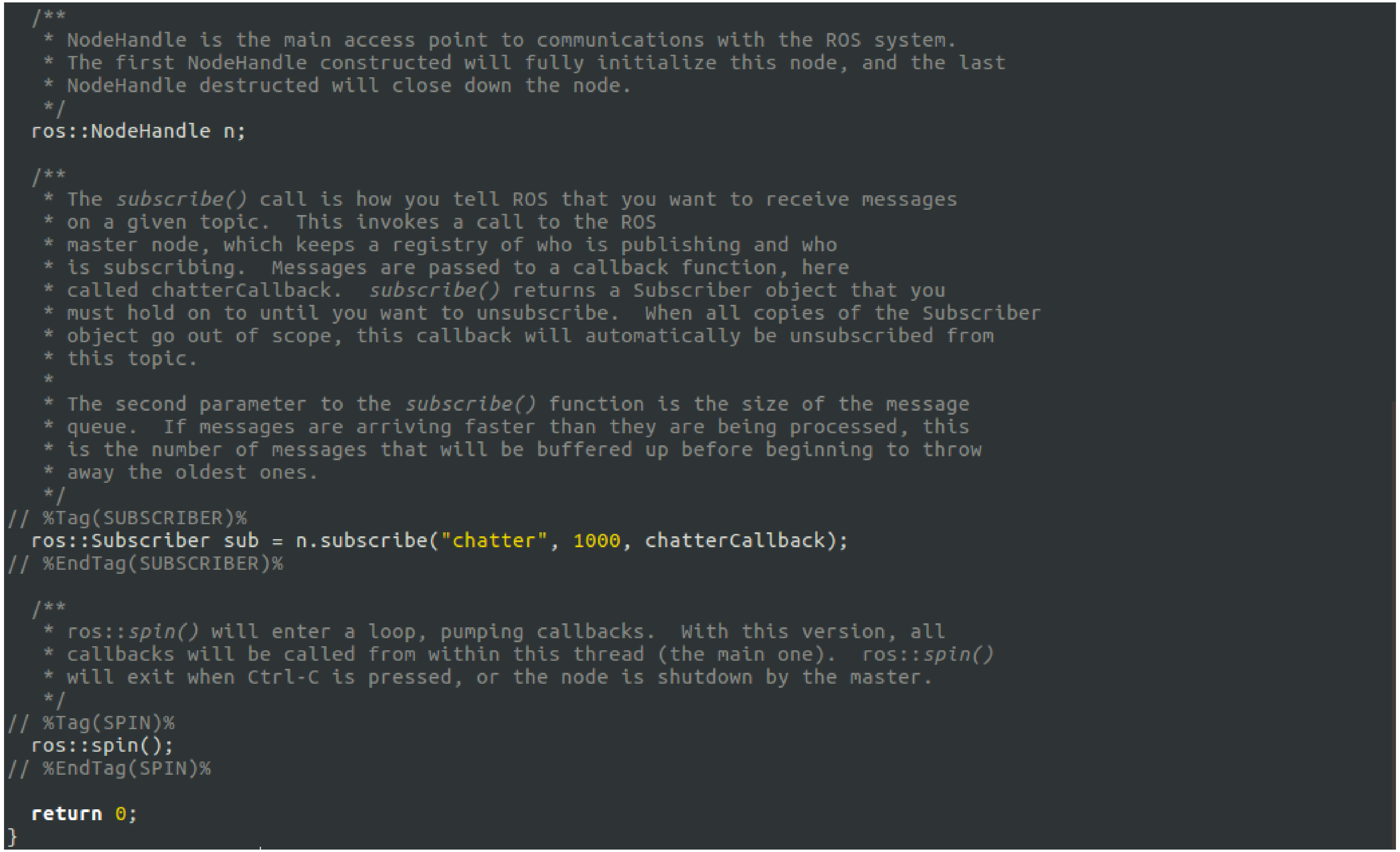
Codice del subscriber (listener.cpp)



```
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
 * AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
 * LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
 * CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
 * SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
 * INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
 * CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
 * ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
 * POSSIBILITY OF SUCH DAMAGE.
 */

// %Tag(FULLTEXT)%
#include "ros/ros.h"
#include "std_msgs/String.h"

/**
 * This tutorial demonstrates simple receipt of messages over the ROS system.
 */
// %Tag(CALLBACK)%
void chatterCallback(const std_msgs::String::ConstPtr& msg)
{
  ROS_INFO("I heard: [%s]", msg->data.c_str());
}
// %EndTag(CALLBACK)%

int main(int argc, char **argv)
{
  /**
   * The ros::init() function needs to see argc and argv so that it can perform
   * any ROS arguments and name remapping that were provided at the command line.
   * For programmatic remappings you can use a different version of init() which takes
   * remappings directly, but for most command-line programs, passing argc and argv is
   * the easiest way to do it.  The third argument to init() is the name of the node.
   *
   * You must call one of the versions of ros::init() before using any other
   * part of the ROS system.
   */
  ros::init(argc, argv, "listener");
```

https://raw.githubusercontent.com/ros/ros_tutorials/kinetic-devel/roscpp_tutorials/listener/listener.cpp

Codice del subscriber (listener.cpp)

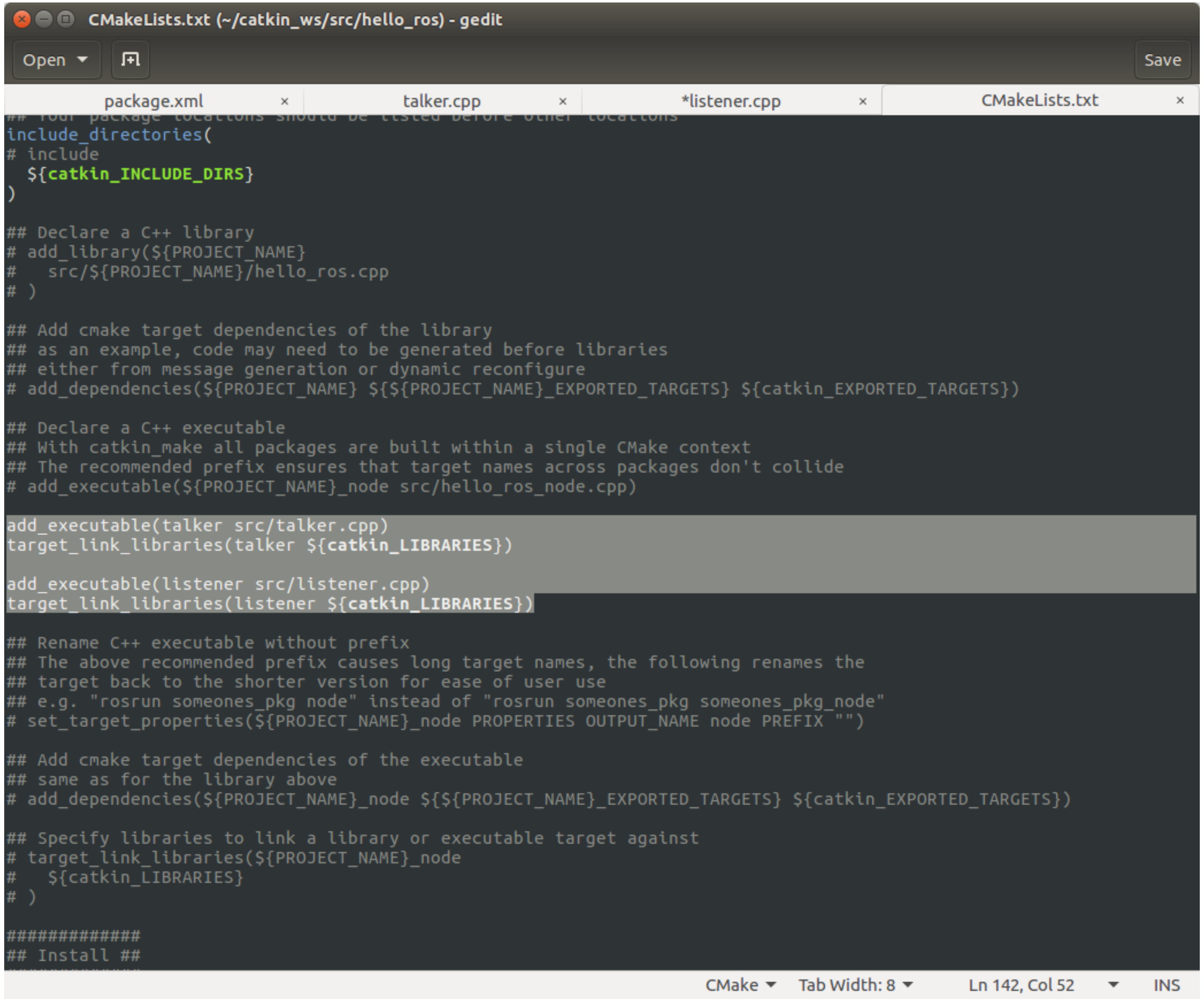


```cpp
  /**
   * NodeHandle is the main access point to communications with the ROS system.
   * The first NodeHandle constructed will fully initialize this node, and the last
   * NodeHandle destructed will close down the node.
   */
  ros::NodeHandle n;

  /**
   * The subscribe() call is how you tell ROS that you want to receive messages
   * on a given topic.  This invokes a call to the ROS
   * master node, which keeps a registry of who is publishing and who
   * is subscribing.  Messages are passed to a callback function, here
   * called chatterCallback.  subscribe() returns a Subscriber object that you
   * must hold on to until you want to unsubscribe.  When all copies of the Subscriber
   * object go out of scope, this callback will automatically be unsubscribed from
   * this topic.
   *
   * The second parameter to the subscribe() function is the size of the message
   * queue.  If messages are arriving faster than they are being processed, this
   * is the number of messages that will be buffered up before beginning to throw
   * away the oldest ones.
   */
// %Tag(SUBSCRIBER)%
  ros::Subscriber sub = n.subscribe("chatter", 1000, chatterCallback);
// %EndTag(SUBSCRIBER)%

  /**
   * ros::spin() will enter a loop, pumping callbacks.  With this version, all
   * callbacks will be called from within this thread (the main one).  ros::spin()
   * will exit when Ctrl-C is pressed, or the node is shutdown by the master.
   */
// %Tag(SPIN)%
  ros::spin();
// %EndTag(SPIN)%

  return 0;
}
```

https://raw.githubusercontent.com/ros/ros_tutorials/kinetic-devel/roscpp_tutorials/listener/listener.cpp

Compiliamo il package hello_ros

Modifichiamo il file CMakeLists.txt in modo da poter compilare il package hello_ros contenente i due nodi talker e listener



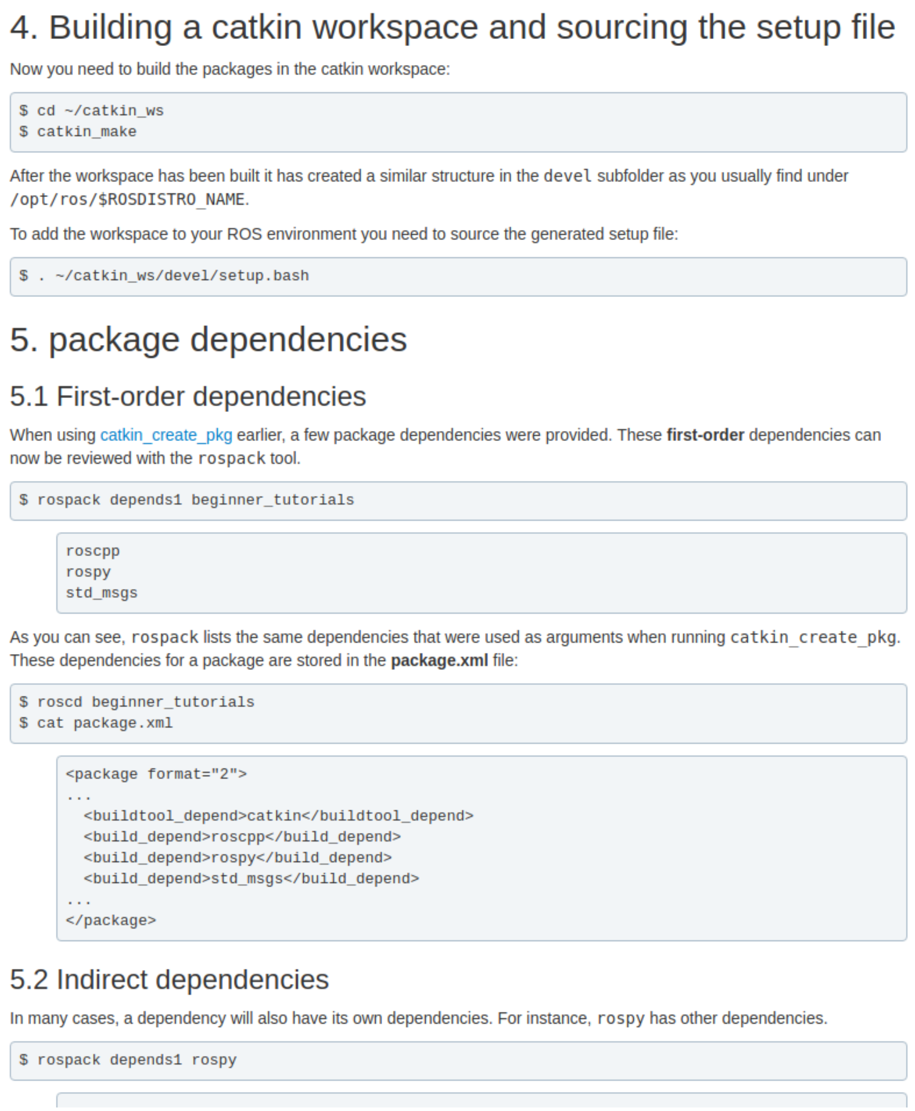
```
## Your package locations should be listed before other locations
include_directories(
# include
  ${catkin_INCLUDE_DIRS}
)

## Declare a C++ library
# add_library(${PROJECT_NAME}
#   src/${PROJECT_NAME}/hello_ros.cpp
# )

## Add cmake target dependencies of the library
## as an example, code may need to be generated before libraries
## either from message generation or dynamic reconfigure
# add_dependencies(${PROJECT_NAME} ${${PROJECT_NAME}_EXPORTED_TARGETS} ${catkin_EXPORTED_TARGETS})

## Declare a C++ executable
## With catkin_make all packages are built within a single CMake context
## The recommended prefix ensures that target names across packages don't collide
# add_executable(${PROJECT_NAME}_node src/hello_ros_node.cpp)

add_executable(talker src/talker.cpp)
target_link_libraries(talker ${catkin_LIBRARIES})

add_executable(listener src/listener.cpp)
target_link_libraries(listener ${catkin_LIBRARIES})

## 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_node"
# 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} ${catkin_EXPORTED_TARGETS})

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

#############
## Install ##
```

CMakeLists.txt

We need the CMakeLists.txt file so that catkin_make, which uses CMake for its more powerful flexibility when building across multiple platforms, builds the package

http://wiki.ros.org/ROS/Tutorials/Creating%20a%20Package%20by%20Hand

Compilazione con catkin_make



4. Building a catkin workspace and sourcing the setup file

Now you need to build the packages in the catkin workspace:

```
$ cd ~/catkin_ws
$ catkin_make
```

After the workspace has been built it has created a similar structure in the `devel` subfolder as you usually find under `/opt/ros/$ROSDISTRO_NAME`.

To add the workspace to your ROS environment you need to source the generated setup file:

```
$ . ~/catkin_ws/devel/setup.bash
```

5. package dependencies

5.1 First-order dependencies

When using catkin_create_pkg earlier, a few package dependencies were provided. These **first-order** dependencies can now be reviewed with the `rospack` tool.

```
$ rospack depends1 beginner_tutorials
```

```
roscpp
rospy
std_msgs
```

As you can see, `rospack` lists the same dependencies that were used as arguments when running `catkin_create_pkg`. These dependencies for a package are stored in the **package.xml** file:

```
$ roscd beginner_tutorials
$ cat package.xml
```

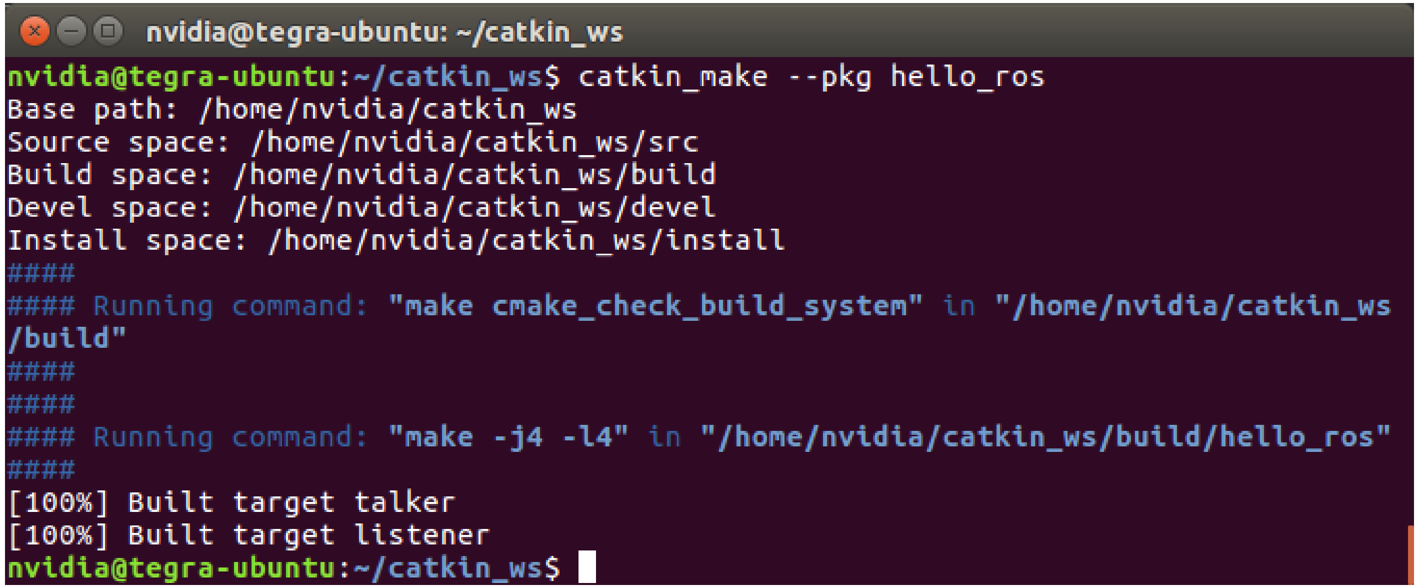
```
<package format="2">
...
  <buildtool_depend>catkin</buildtool_depend>
  <build_depend>roscpp</build_depend>
  <build_depend>rospy</build_depend>
  <build_depend>std_msgs</build_depend>
...
</package>
```

5.2 Indirect dependencies

In many cases, a dependency will also have its own dependencies. For instance, `rospy` has other dependencies.

```
$ rospack depends1 rospy
```

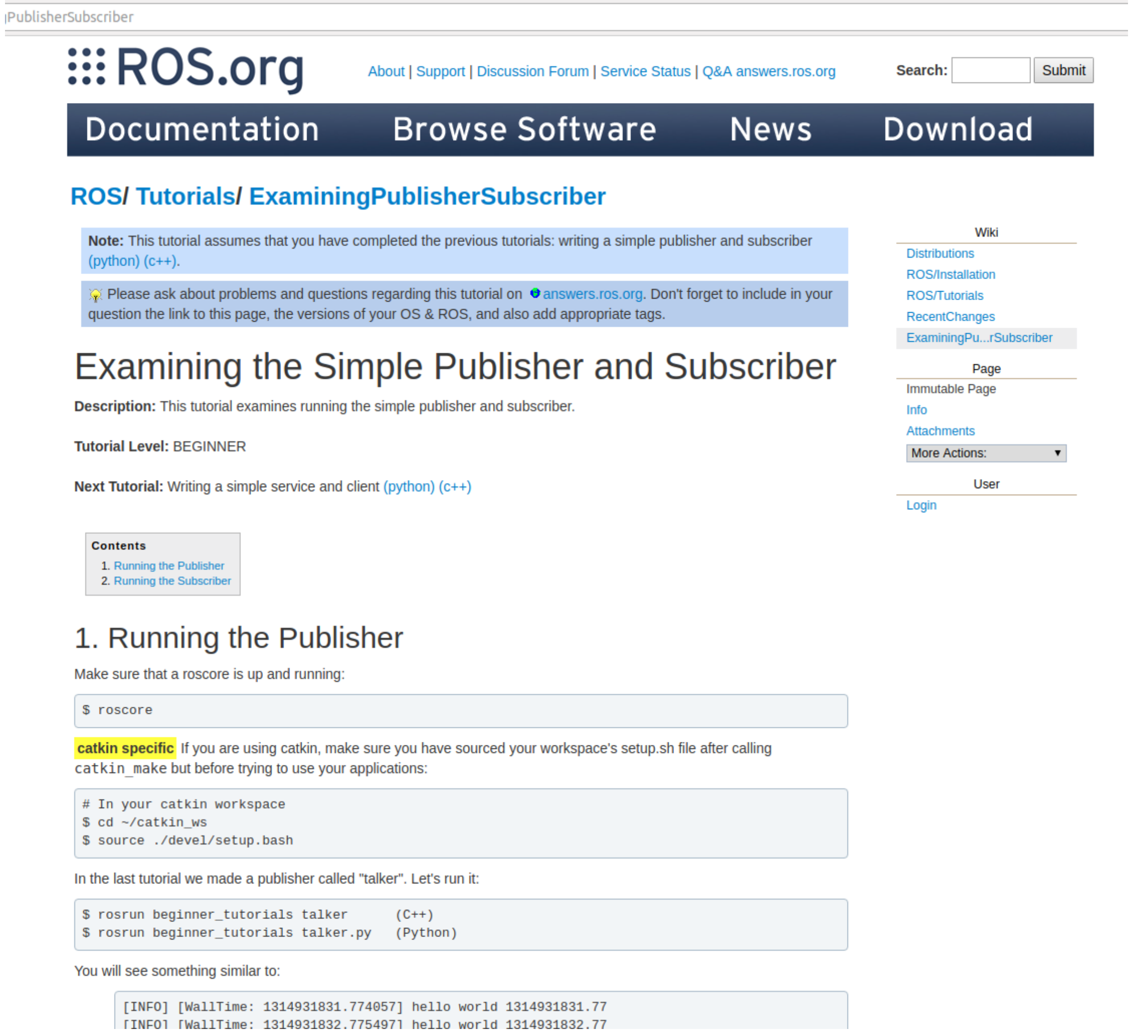
```
catkin_make --pkg hello_ros
```



```
nvidia@tegra-ubuntu:~/catkin_ws$ catkin_make --pkg hello_ros
Base path: /home/nvidia/catkin_ws
Source space: /home/nvidia/catkin_ws/src
Build space: /home/nvidia/catkin_ws/build
Devel space: /home/nvidia/catkin_ws/devel
Install space: /home/nvidia/catkin_ws/install
####
#### Running command: "make cmake_check_build_system" in "/home/nvidia/catkin_ws
/build"
####
####
#### Running command: "make -j4 -l4" in "/home/nvidia/catkin_ws/build/hello_ros"
####
[100%] Built target talker
[100%] Built target listener
nvidia@tegra-ubuntu:~/catkin_ws$
```

http://wiki.ros.org/ROS/Tutorials/catkin/CreatingPackage

Esecuzione del nodo talker



ROS/ Tutorials/ ExaminingPublisherSubscriber

Note: This tutorial assumes that you have completed the previous tutorials: writing a simple publisher and subscriber (python) (c++).

Please ask about problems and questions regarding this tutorial on answers.ros.org. Don't forget to include in your question the link to this page, the versions of your OS & ROS, and also add appropriate tags.

Examining the Simple Publisher and Subscriber

Description: This tutorial examines running the simple publisher and subscriber.

Tutorial Level: BEGINNER

Next Tutorial: Writing a simple service and client (python) (c++)

1. Running the Publisher

Make sure that a roscore is up and running:

```
$ roscore
```

catkin specific If you are using catkin, make sure you have sourced your workspace's setup.sh file after calling `catkin_make` but before trying to use your applications:

```
# In your catkin workspace
$ cd ~/catkin_ws
$ source ./devel/setup.bash
```

In the last tutorial we made a publisher called "talker". Let's run it:

```
$ rosrun beginner_tutorials talker      (C++)
$ rosrun beginner_tutorials talker.py   (Python)
```

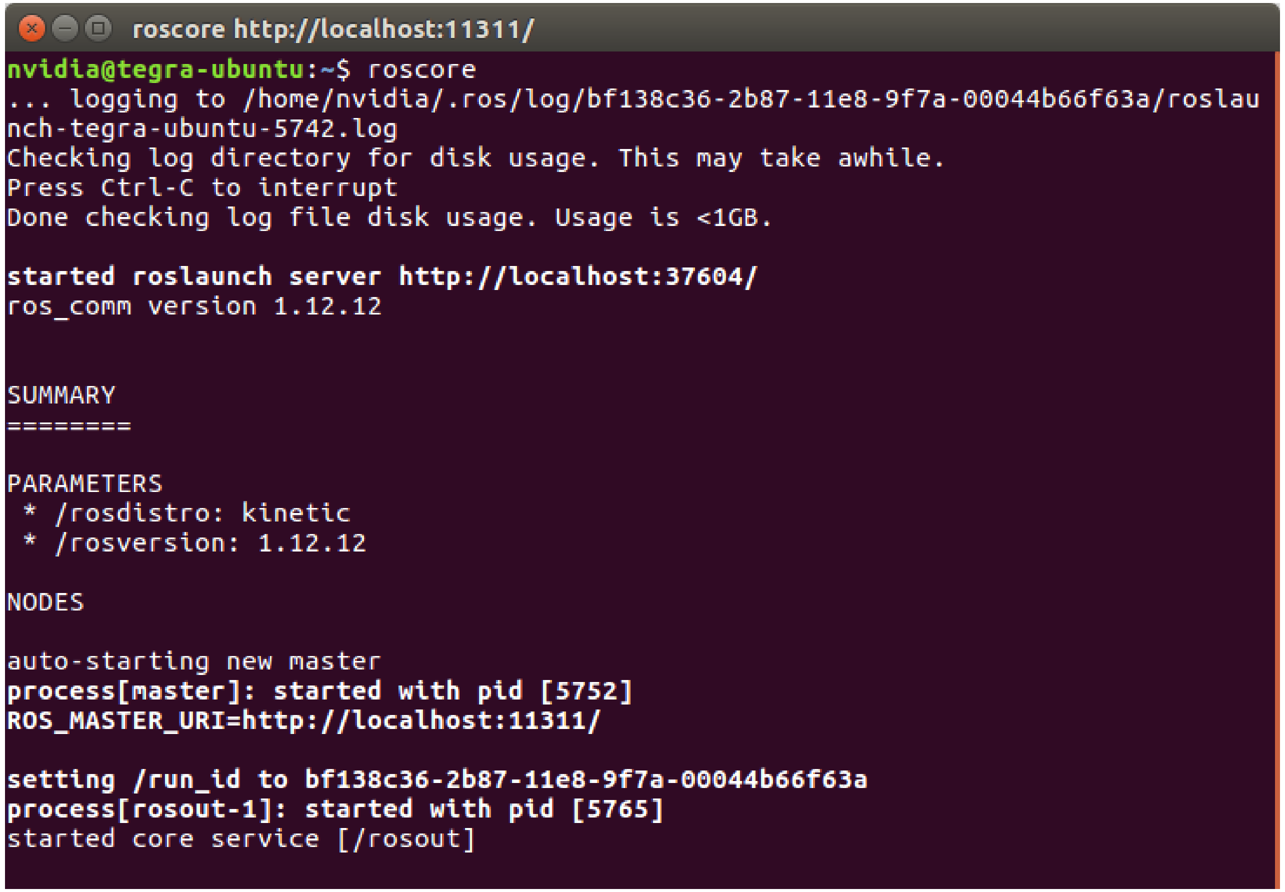
You will see something similar to:

```
[INFO] [WallTime: 1314931831.774057] hello world 1314931831.77
[INFO] [WallTime: 1314931832.775497] hello world 1314931832.77
```

http://wiki.ros.org/ROS/Tutorials/ExaminingPublisherSubscriber

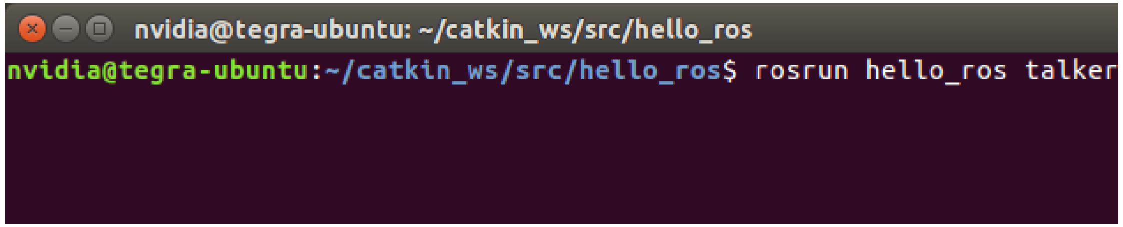
roscore + rosrun

Apriamo un terminale e lanciamo `roscore`



```
nvidia@tegra-ubuntu:~$ roscore
... logging to /home/nvidia/.ros/log/bf138c36-2b87-11e8-9f7a-00044b66f63a/roslau
nch-tegra-ubuntu-5742.log
Checking log directory for disk usage. This may take awhile.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://localhost:37604/
ros_comm version 1.12.12


SUMMARY
========

PARAMETERS
 * /rosdistro: kinetic
 * /rosversion: 1.12.12

NODES

auto-starting new master
process[master]: started with pid [5752]
ROS_MASTER_URI=http://localhost:11311/

setting /run_id to bf138c36-2b87-11e8-9f7a-00044b66f63a
process[rosout-1]: started with pid [5765]
started core service [/rosout]
```

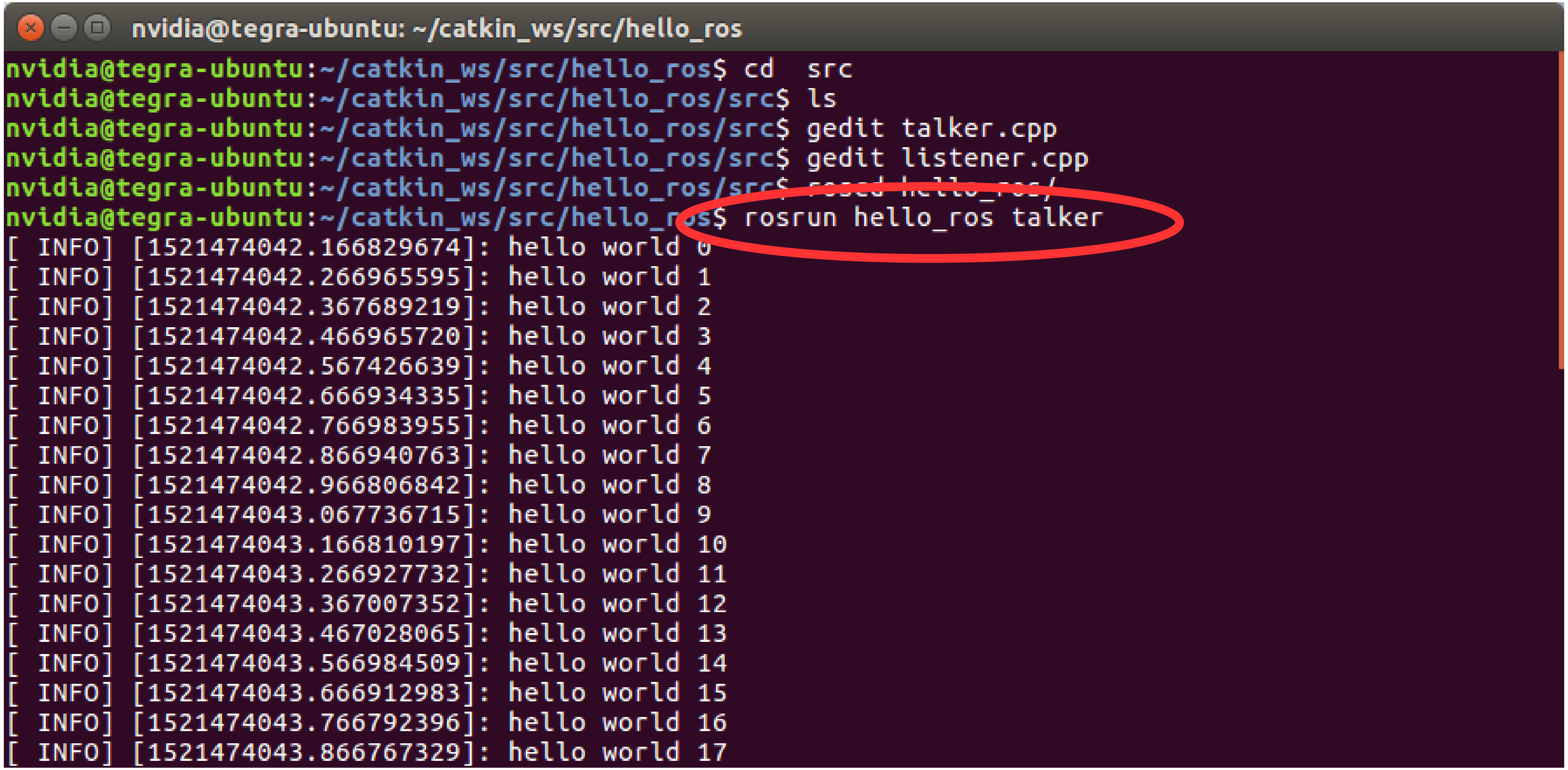
Apriamo un secondo terminale e lanciamo `rosrun hello_ros talker`



```
nvidia@tegra-ubuntu:~/catkin_ws/src/hello_ros$ rosrun hello_ros talker
```

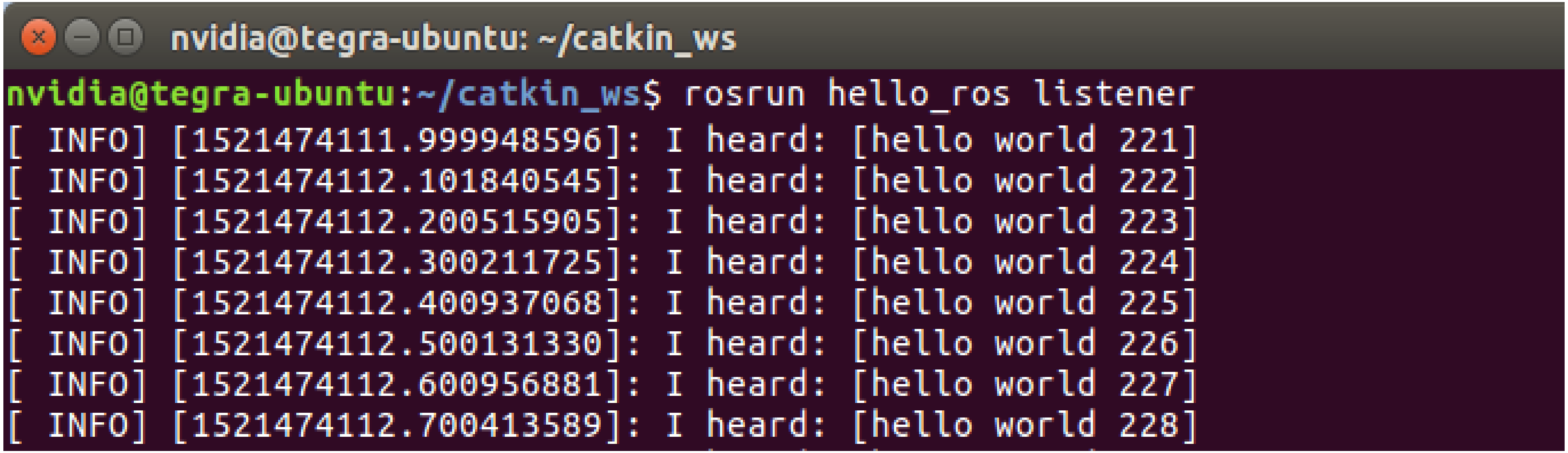
Cosa accade?

Esecuzione del nodo talker



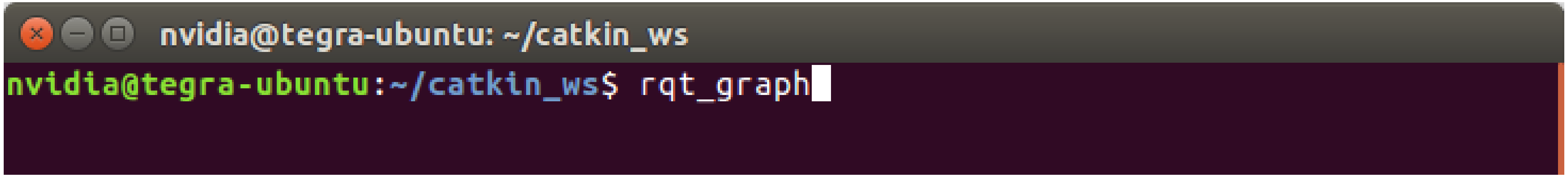
```
nvidia@tegra-ubuntu:~/catkin_ws/src/hello_ros$ cd  src
nvidia@tegra-ubuntu:~/catkin_ws/src/hello_ros/src$ ls
nvidia@tegra-ubuntu:~/catkin_ws/src/hello_ros/src$ gedit talker.cpp
nvidia@tegra-ubuntu:~/catkin_ws/src/hello_ros/src$ gedit listener.cpp
nvidia@tegra-ubuntu:~/catkin_ws/src/hello_ros/src$ roscd hello_ros/
nvidia@tegra-ubuntu:~/catkin_ws/src/hello_ros$ rosrun hello_ros talker
[ INFO] [1521474042.166829674]: hello world 0
[ INFO] [1521474042.266965595]: hello world 1
[ INFO] [1521474042.367689219]: hello world 2
[ INFO] [1521474042.466965720]: hello world 3
[ INFO] [1521474042.567426639]: hello world 4
[ INFO] [1521474042.666934335]: hello world 5
[ INFO] [1521474042.766983955]: hello world 6
[ INFO] [1521474042.866940763]: hello world 7
[ INFO] [1521474042.966806842]: hello world 8
[ INFO] [1521474043.067736715]: hello world 9
[ INFO] [1521474043.166810197]: hello world 10
[ INFO] [1521474043.266927732]: hello world 11
[ INFO] [1521474043.367007352]: hello world 12
[ INFO] [1521474043.467028065]: hello world 13
[ INFO] [1521474043.566984509]: hello world 14
[ INFO] [1521474043.666912983]: hello world 15
[ INFO] [1521474043.766792396]: hello world 16
[ INFO] [1521474043.866767329]: hello world 17
```

Esecuzione del nodo listener



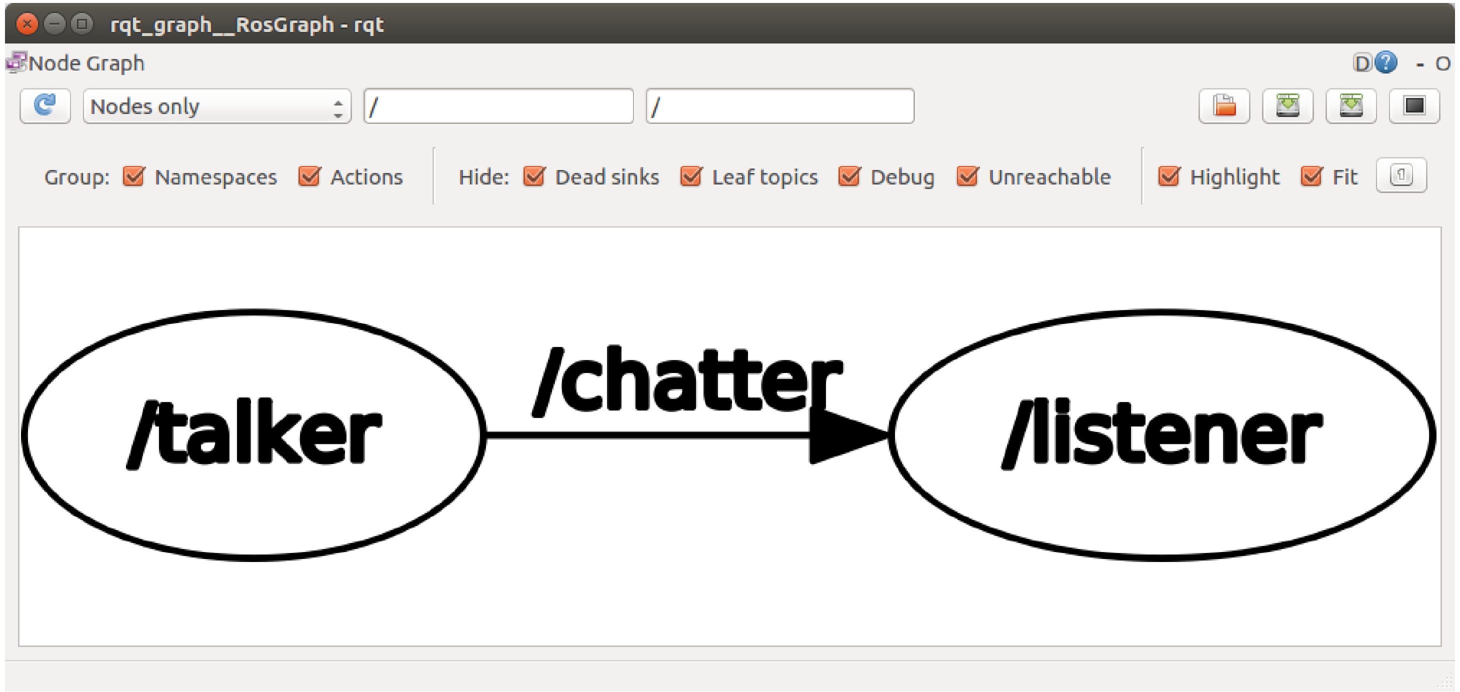
```
nvidia@tegra-ubuntu:~/catkin_ws$ rosrun hello_ros listener
[ INFO] [1521474111.999948596]: I heard: [hello world 221]
[ INFO] [1521474112.101840545]: I heard: [hello world 222]
[ INFO] [1521474112.200515905]: I heard: [hello world 223]
[ INFO] [1521474112.300211725]: I heard: [hello world 224]
[ INFO] [1521474112.400937068]: I heard: [hello world 225]
[ INFO] [1521474112.500131330]: I heard: [hello world 226]
[ INFO] [1521474112.600956881]: I heard: [hello world 227]
[ INFO] [1521474112.700413589]: I heard: [hello world 228]
```

rqt_graph



nvidia@tegra-ubuntu: ~/catkin_ws

```
nvidia@tegra-ubuntu:~/catkin_ws$ rqt_graph
```



rqt_graph__RosGraph - rqt

Node Graph

Nodes only · / · /

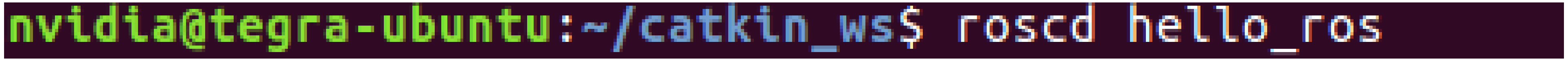
Group: ☑ Namespaces ☑ Actions

Hide: ☑ Dead sinks ☑ Leaf topics ☑ Debug ☑ Unreachable

☑ Highlight ☑ Fit

Aggiorniamo il repository locale

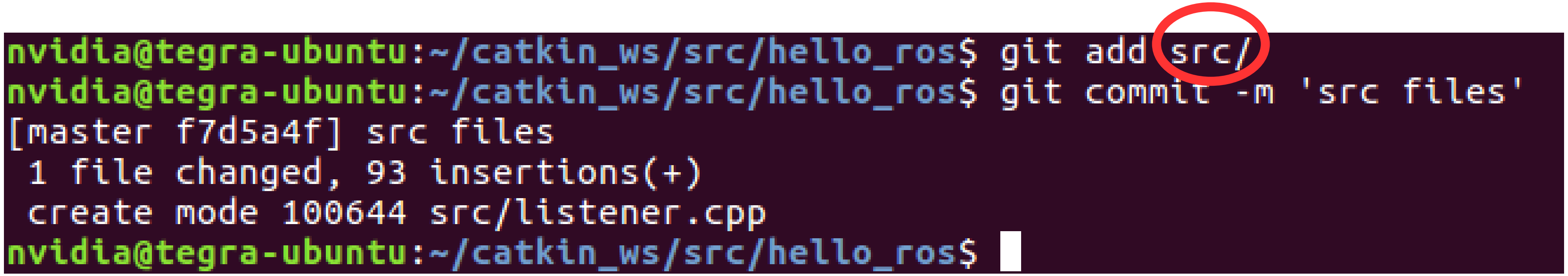
Con roscd possiamo navigare nel filesystem per portarci nella directory del nostro package



```
nvidia@tegra-ubuntu:~/catkin_ws$ roscd hello_ros
```

Aggiorniamo il repository locale con la cartella src

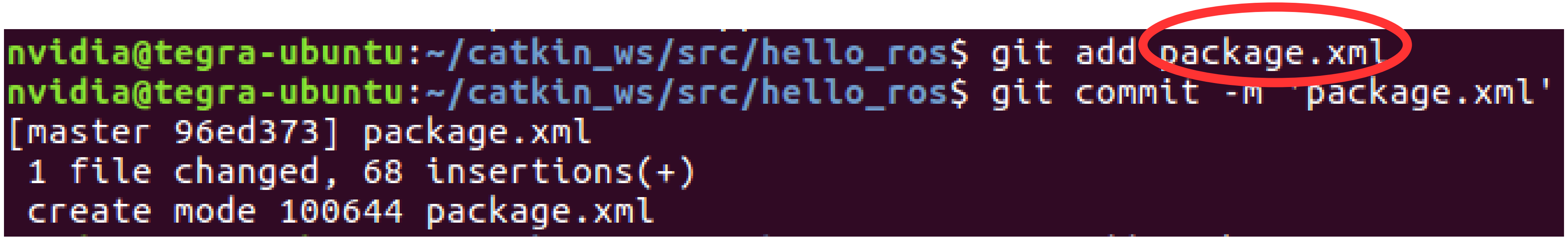
```
git add
git commit
```



```
nvidia@tegra-ubuntu:~/catkin_ws/src/hello_ros$ git add src/
nvidia@tegra-ubuntu:~/catkin_ws/src/hello_ros$ git commit -m 'src files'
[master f7d5a4f] src files
 1 file changed, 93 insertions(+)
 create mode 100644 src/listener.cpp
nvidia@tegra-ubuntu:~/catkin_ws/src/hello_ros$
```

Aggiorniamo il repository locale (package.xml)

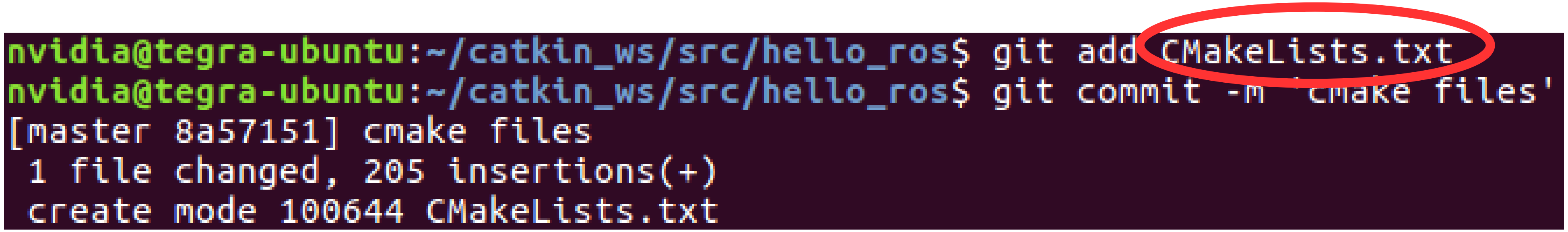
```
git add
git commit
```



```
nvidia@tegra-ubuntu:~/catkin_ws/src/hello_ros$ git add package.xml
nvidia@tegra-ubuntu:~/catkin_ws/src/hello_ros$ git commit -m 'package.xml'
[master 96ed373] package.xml
 1 file changed, 68 insertions(+)
 create mode 100644 package.xml
```

Aggiorniamo il repository locale (CMakeLists.txt)

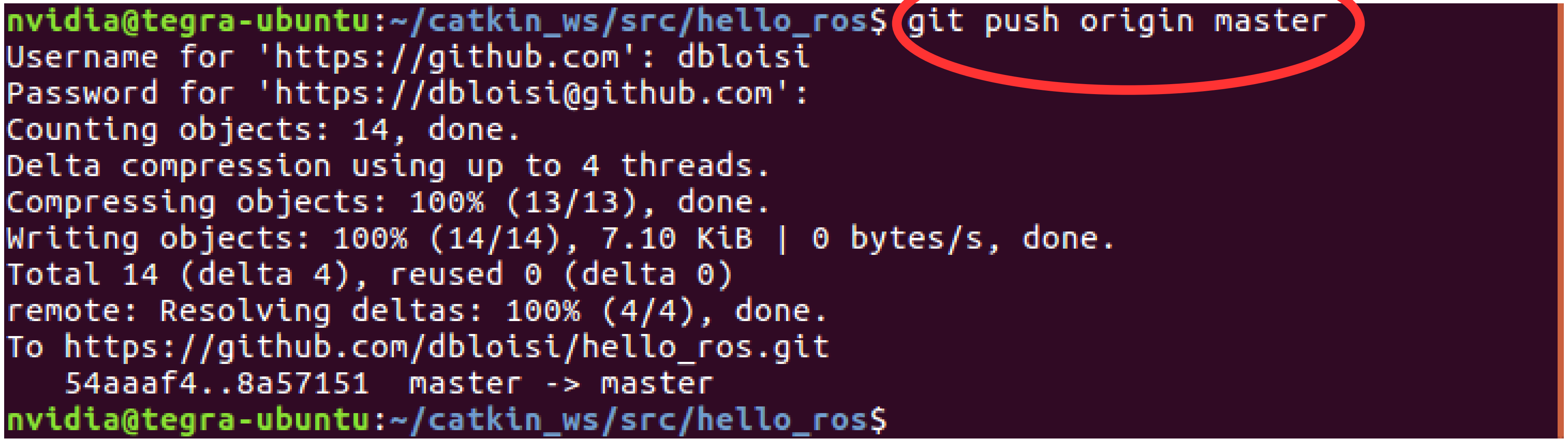
```
git add
git commit
```



```
nvidia@tegra-ubuntu:~/catkin_ws/src/hello_ros$ git add CMakeLists.txt
nvidia@tegra-ubuntu:~/catkin_ws/src/hello_ros$ git commit -m 'cmake files'
[master 8a57151] cmake files
 1 file changed, 205 insertions(+)
 create mode 100644 CMakeLists.txt
```

Aggiorniamo il repository remoto

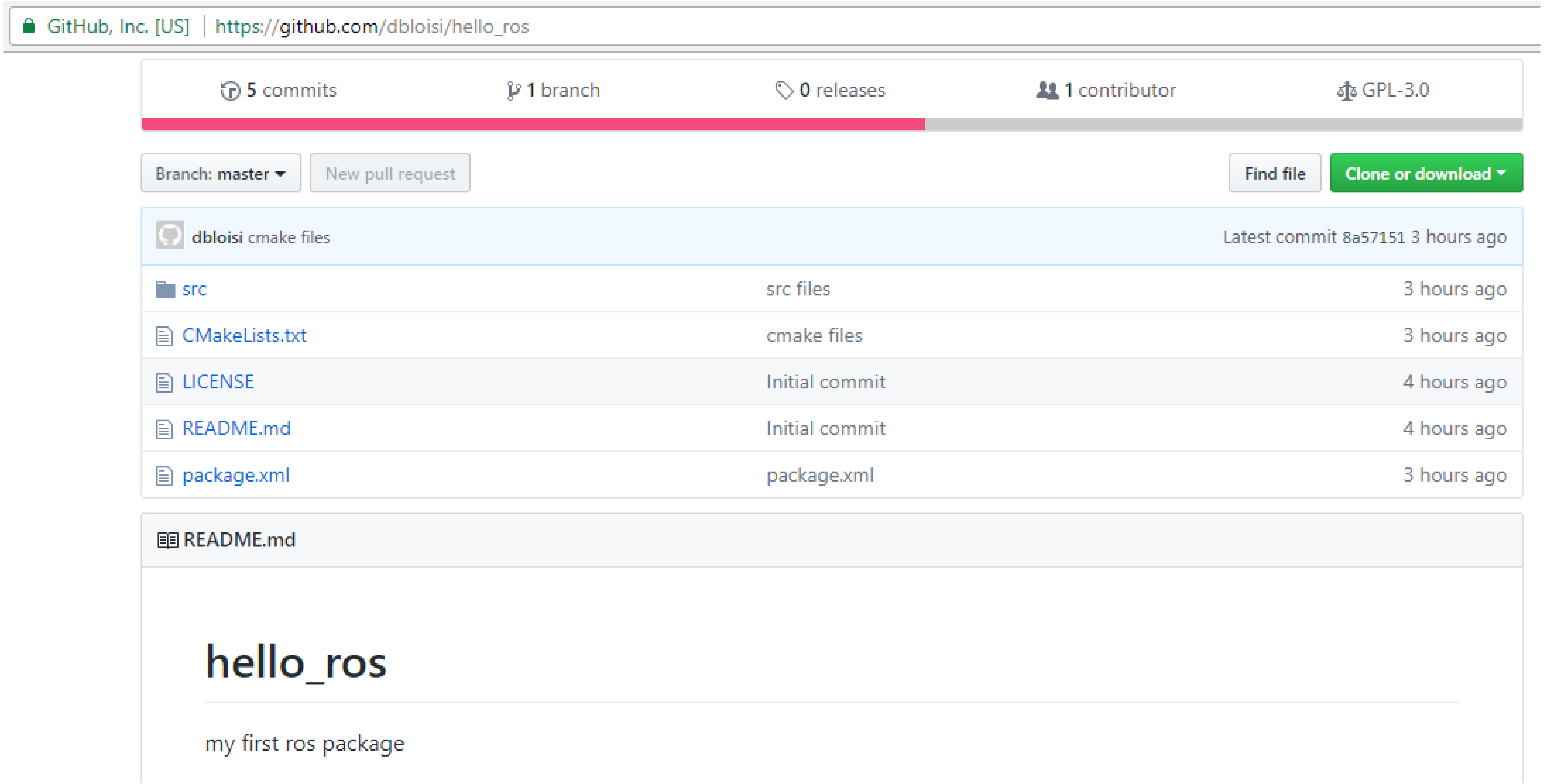
`git push`



```
nvidia@tegra-ubuntu:~/catkin_ws/src/hello_ros$ git push origin master
Username for 'https://github.com': dbloisi
Password for 'https://dbloisi@github.com':
Counting objects: 14, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (13/13), done.
Writing objects: 100% (14/14), 7.10 KiB | 0 bytes/s, done.
Total 14 (delta 4), reused 0 (delta 0)
remote: Resolving deltas: 100% (4/4), done.
To https://github.com/dbloisi/hello_ros.git
   54aaaf4..8a57151  master -> master
nvidia@tegra-ubuntu:~/catkin_ws/src/hello_ros$
```

Verranno richieste le credenziali di accesso (username e password) per il server git

Aggiorniamo il repository remoto



dbloisi cmake files — Latest commit 8a57151 3 hours ago

src	src files	3 hours ago
CMakeLists.txt	cmake files	3 hours ago
LICENSE	Initial commit	4 hours ago
README.md	Initial commit	4 hours ago
package.xml	package.xml	3 hours ago

README.md

hello_ros

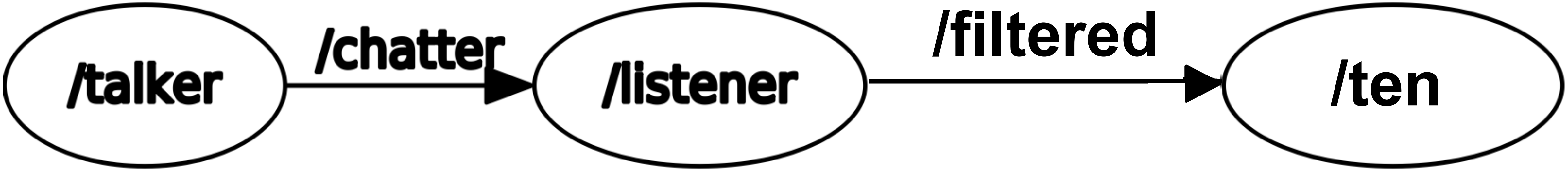
my first ros package

Esercitazione

1. Creare un account su un server git (es. GitHub, BitBucket, GitLab)
2. Creare un repository denominato my_hello_ros
3. Creare un package my_hello_ros contenente i nodi talker e listener
4. Caricare il codice sul proprio repository

Esercitazione

5. Modificare il codice del listener in modo che pubblichi a sua volta un messaggio dopo aver ascoltato 10 messaggi provenienti dal talker
6. Creare un nuovo nodo ten che ascolti i messaggi del listener e li stampi a video
7. Aggiornare il repository remoto

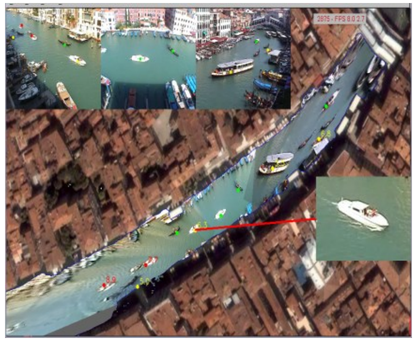


UNIVERSITÀ di VERONA | Dipartimento di INFORMATICA

Corso di Laboratorio Ciberfisico

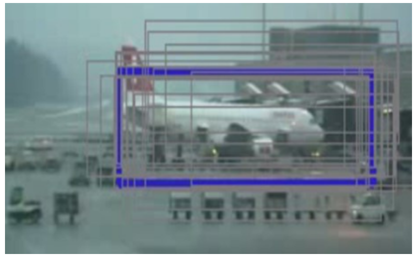
Modulo di Robot Programming with ROS

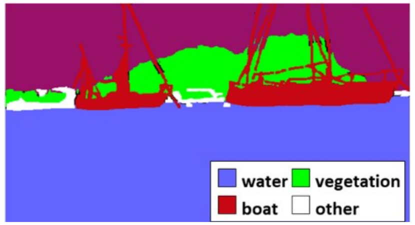
git + ROS



Docente:

Domenico Daniele Bloisi





Marzo 2018