ROS

# 安装：

<http://wiki.ros.org/kinetic/Installation/Ubuntu>

# commands

***roswtf*** *# 检测环境变量、安装的文件以及运行的节点*

***rospack list***

***rospack find package-name***

***rosls package-name***

***roscd package-name***

***rosnode list***

***rosnode info node-name***

***rosnode kill node-name***

***rosnode cleanup***

***rostopic list***

***rostopic echo topic-name #*** *rostopic echo /turtle1/cmd\_vel*

***rostopic hz topic-name***

***rostopic bw topic-name***

***rostopic info topic-name***

***rostopic pub –r rate-in-hz topic-name message-type message-content***

***#****rostopic pub –r 1 /turtle1/cmd\_vel geometry\_msgs/Twist ’[0,0,0]’ ’[0,0,1]’*

***rosmsg show message-type-name***

# ROS turtle\_sample

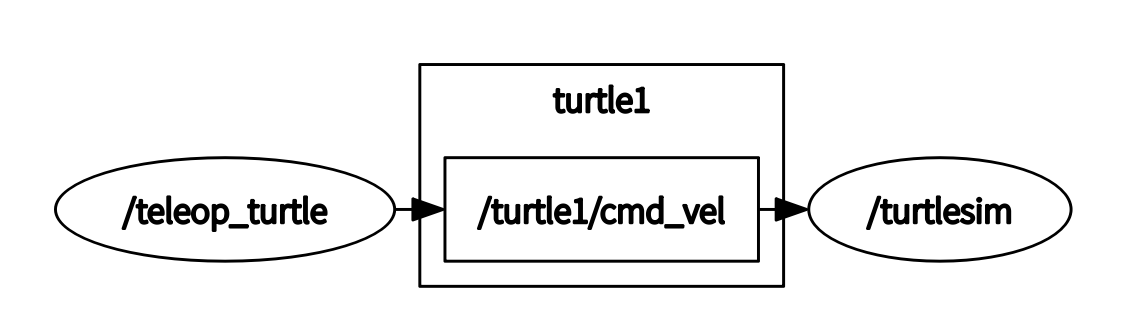
*roscore #启动节点管理器*

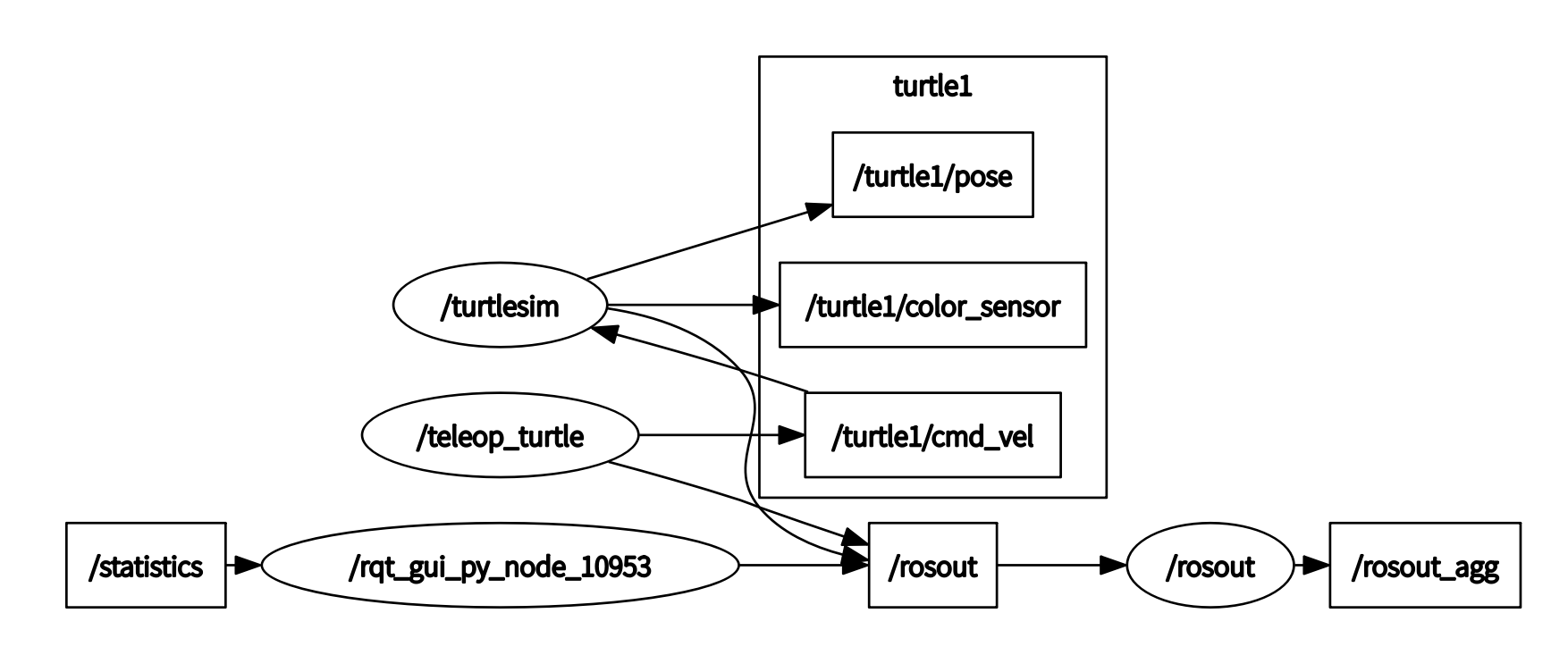
*rosrun turtlesim turtlesim\_node*

*rosrun turtlesim turtle\_teleop\_key*

*#rosrun package-name executable-name*

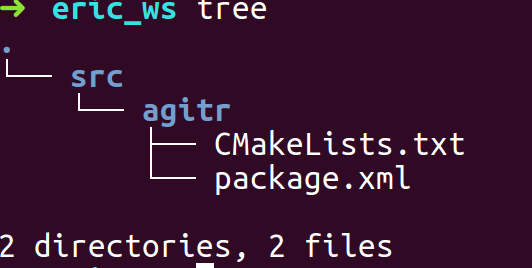
*#rosrun package-name executable-name \_\_name:=node-name*

**

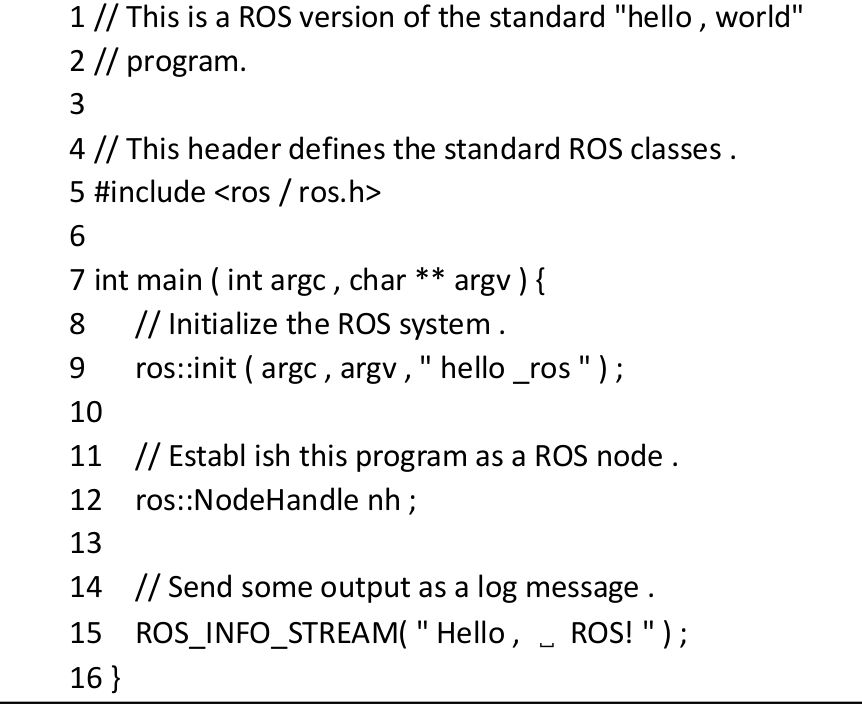
**

# 开发程序

* 创建工作区：mkdir ws\_name、mkdir ws\_name/src
  + 创建功能包：catkin\_create\_pkg package-name #catkin\_create\_pkg agitr

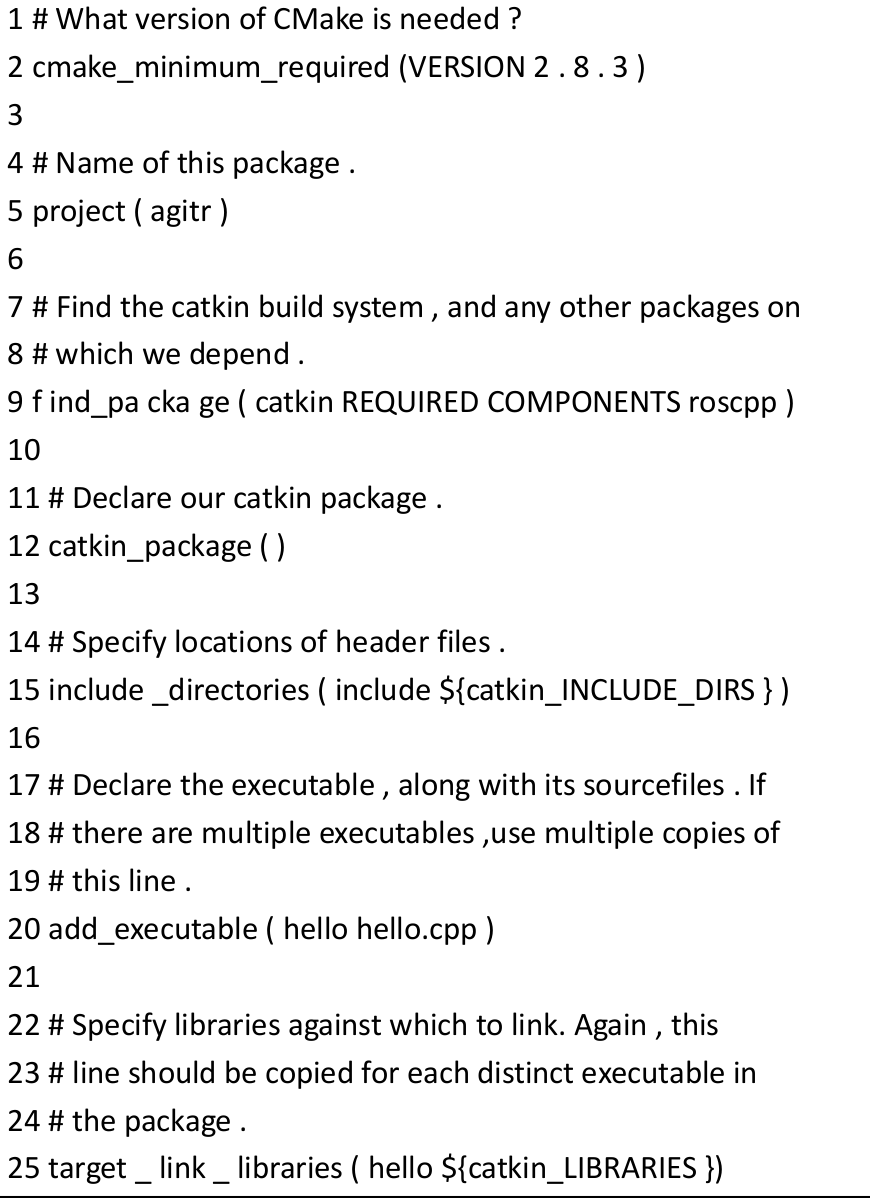


* + 编写代码

**

*创建此对象会将程序注册为ROS节点管理器的节点*

* 编译程序



* 编译工作区

catkin\_make

* Sourcing setup.bash

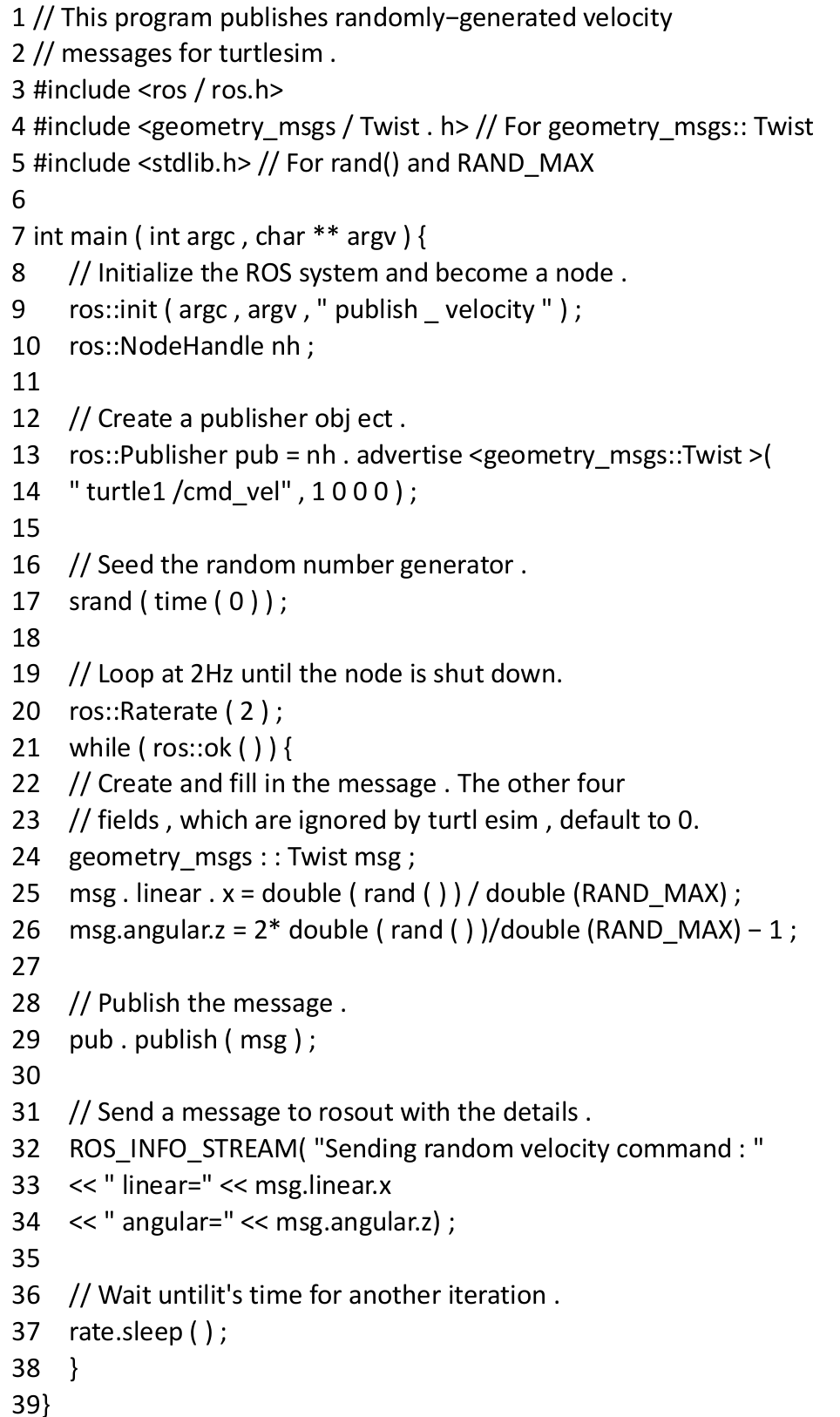
source devel/setup.bash

#*设置了若干环境变量,从而使 ROS 能够找到你创建的功能包和新生成的可执行文件*

* 执行程序

rosrun agitr hello

## 发布消息



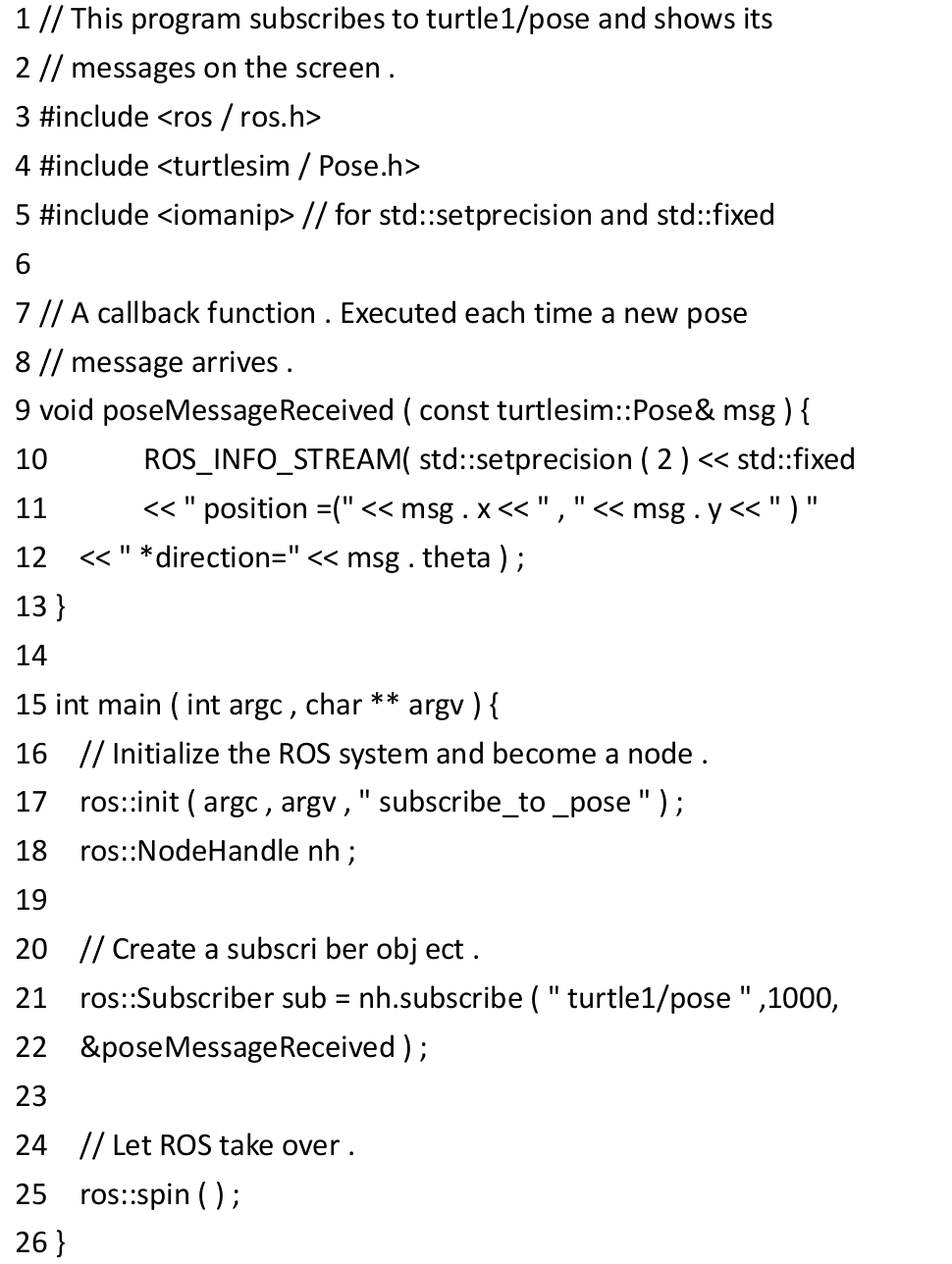
创建发布者：ros::Publisher pub = node\_handle.advertise<message\_type>(

topic\_name, queue\_size);

发布消息

## 订阅消息

* ROS\_\*\*\*\_STREAM #message 可处理 C++中标准输出流(ostream)中的各种表达式如 std::cout
* ROS\_\*\*\* #类似格式输出函数printf
* ROS\_\*\*\*\_STREAM\_ONCE
* ROS\_\*\*\*\_STREAM\_THROTTLE(interval, message) # 频率受控
* \*\*\*：DEBUG/INFO/WARN/ERROR/FATAL
* 输出到控制台



ros::Subscriber sub =node\_handle.subscribe

(topic\_name,queue\_size, pointer\_to\_callback\_function);

程序除了响应回调函数，没有其他重复性工作要做，那么使用 ros::spin(); 否则，合理的选择是写一个循环,做其他需要做的事情,并且周期性地调用 ros::spinOnce()来处理回调

while(ros::ok( ))

{

ros::spinOnce();

}

## 日志

### 控制台

command > file #只重定向标准输出,不包括标准错误

command &> file #将所有日志消息重定向到同一个文件

注：两种流的缓存方式不同,可能导致消息不按照顺序出现

stdbuf –oL command &> file

注：ROS 会在输出中插入人类和软件无法理解ANSI 颜色编码。less –r file

格式化控制台消息：

${severity} ${time} ${message} ${file} ${line} ${function} ${node}

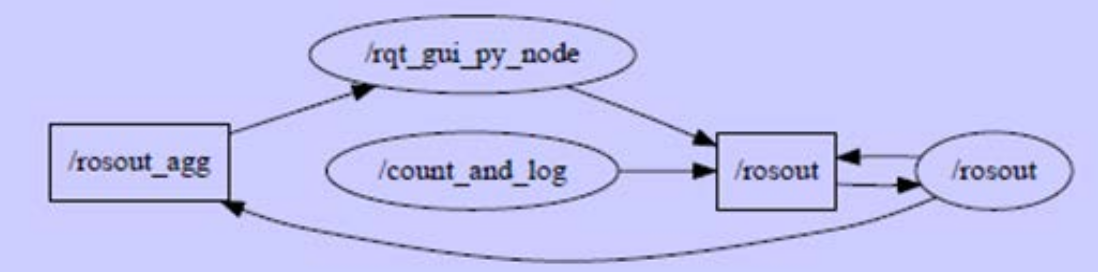
### rosout

每一个日志消息都被发布到话题/rosout 上。该话题的消息类型是 rosgraph\_msgs/Log

查看消息：rostopic echo /rosout 或者 rqt\_console

rqt\_console 订阅的是/rosout\_agg,而不是/rosout

rosout 是唯一一个订阅/rosout 的节点,并且是/rosout\_agg 话题的唯一发布者。这样,调试工具通过订阅/rosout\_agga 就可以获得完整的日志消息流,而不需系统中每一个节点做额外工作



### 日志文件

### ~/.ros/log/run\_id/rosout.log

### 运行标识码(run\_id)是一个通用唯一识别码(UUID)，区分来自不同 ROS 会话的日志文件。

### roscore 生成的输出：setting /run\_id to **run\_id**

### rosparam get /run\_id

### 清除日志文件：

### rosclean check

### rosclean purge

### 启用和禁用日志

### 缺省日志级别info

### rosservice call /node-name/set\_logger\_level ros.package-name level

### *rosservice call /count\_and\_log/set\_logger\_level ros.agitr DEBUG #在节点启动后调用*

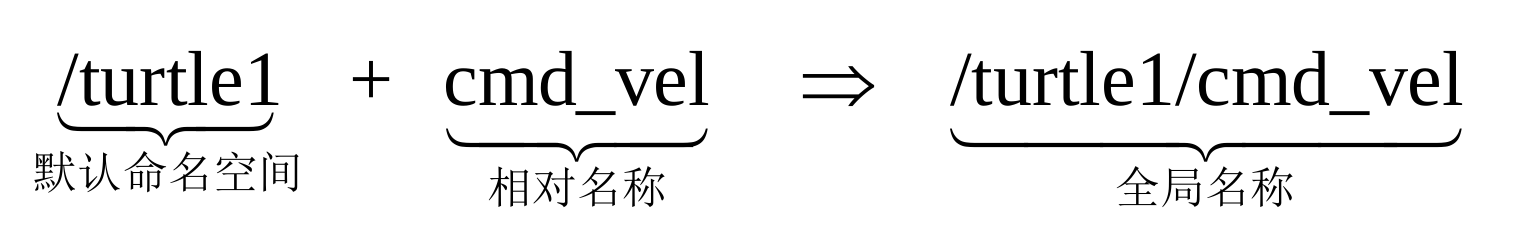
### rqt\_logger\_level

## 计算图源命名

### 全局名称

### 节点、话题、服务和参数统称为计算图源，而每个计算图源由一个叫计算图源名称(graph resource name)的短字符串标识

### 相对名称

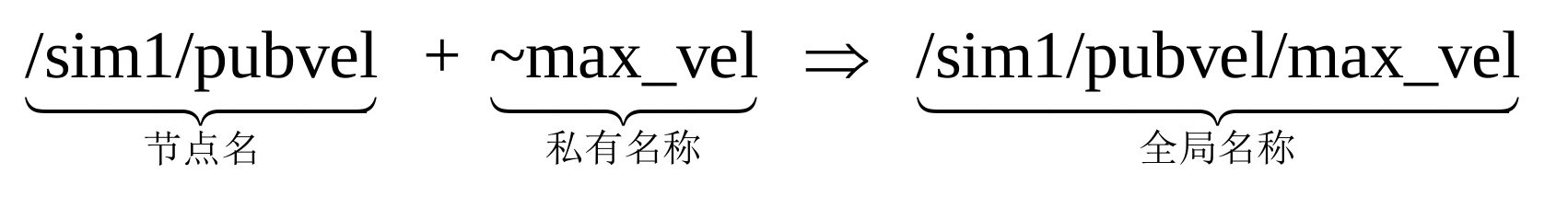


### 设置默认命名空间

* 调用 ros::init 的所有 C++程序接受\_\_ns 的命令行参数
* 环境变量Export ROS\_NAMESPACE=default-namespace，只有当没有其他由\_ \_ns 参数指定的默认命名空间时才有效

### 私有名称

### 私有名称不用当前默认命名空间，用的它们节点名称作为命名空间

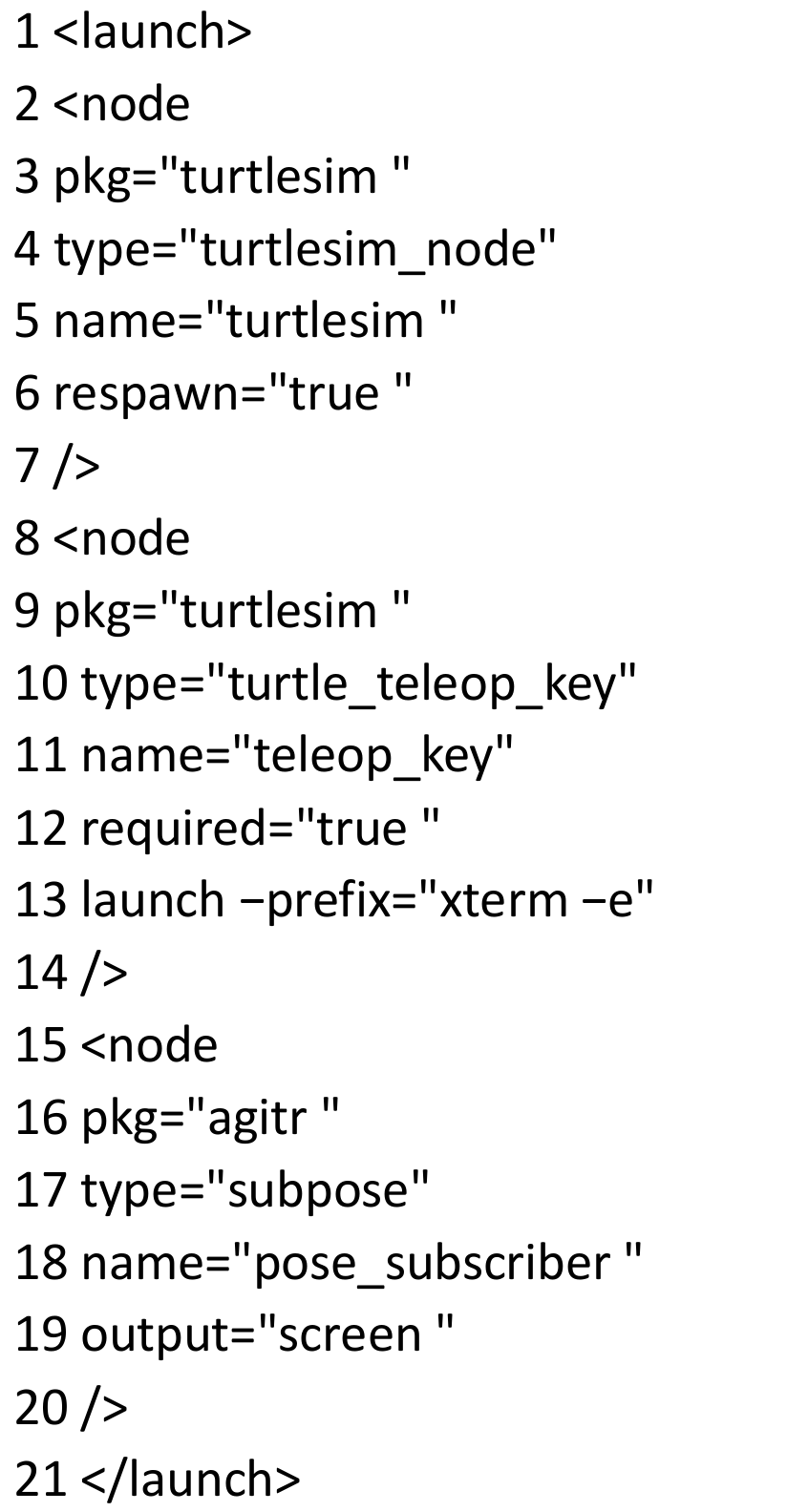


* 每个节点内部都有这样一些资源，只与本节点有关，不会与其他节点打交道，这些资源就可以使用私有名称
* 私有名称常常用于参数，也用于管理一个节点运算的服务
* 也对于其他节点来讲,只要知道私有名称解析后的全局名称，都可以通过其全局名称访问这些计算图源。

### 匿名名称

* ROS 还提供了另一种被称为匿名名称的命名机制，一般用于为节点命名
* 当节点调用 ros::init 方法时可以请求一个自动分配的唯一名称
* ros::init(argc, argv, base\_name, **ros::init\_options::AnonymousName**)在节点的基本名称后面追加某个额外的文本，以确保节点的名字是唯一

## 启动文件

roslaunch package-name launch-file-name

roslaunch agitr example.launch

-v 用来观察 roslaunch 如何解释你的启动文件

### 启动文件位置

### 最简单的方法是把启动文件直接存储在功能包的根目录中

### 当查找启动文件的时候,roslaunch 工具会同时搜索每个功能包目录的子目录。包括ROS 核心包在内的很多功能包都是利用这一特性,将所有启动文件统一存放在一个子目录中,**该子目录通常取名为 launch**

### **启动文件基本元素**

### <launch>

### <node

### pkg=”package-name”

### type=”executable-name”

### name=”node-name” #匿名name=”$(anon base\_name)”

### />