Installing and Configuring Your ROS Environment (เฉพาะ ROS 1เท่านั้น)

การติดตั้ง ROS และการสร้าง environment

1. ติดตั้ง ROS (http://wiki.ros.org/ROS/Installation)
2. การจัดการกับ environment

   ในขณะที่ติดตั้ง ROS คุณจะได้รับการแจ้งเตือนว่าให้ทำการเลือก source ไฟล์ .*sh ขึ้นมา 1ไฟล์ หรือทำการ source ที่ shell startup script ของเรา สิ่งนี้จำเป็นที่จะต้องทำแพราะ ROS อาศัยการแบ่งพื้นที่ส่วนหนึ่งจาก shell environment การทำการ source แบบนี้ ทำให้การใช้งาน ROS ที่มี Version ต่างกัน และการใช้งานกับ Package ที่ Version แตกต่างกันสามารถใช้งานได้ง่ายขึ้น

   ถ้าผู้เรียนเจอปัญหาในการใช้ หรือ ค้นหา ROS package ผู้เรียนจะต้องตรวจให้แน่ใจว่าได้ทำการตั้งค่า environment ไว้อย่างถูกต้องไหม (จากการเช็ค ตัวแปรเกี่ยวกับ environment เช่น ROS_PACKAGE และ ROS_PACKAGE_PATH ว่าถูกเซตไว้แล้วหรือยัง?)

   สามารถเช็คได้จากการพิมพ์โค้ดด้านล่างไปที่ terminal

```
$ printenv | grep ROS
```

ถ้าหากยังไม่มีการตั้งค่า environment แสดงว่า ผู้เรียนจะต้องทำการ source ไฟล์ setup .*sh บางไฟล์เสียก่อน

ไฟล์ Environment setup นั้นถูกสร้างขึ้นแล้ว เพียงแต่มันไม่สามารถทำงานได้เนื่องจากไฟล์ไม่อยู่ในที่เดียวกัน ดังเช่นสถานการณ์ด้านล่างนี้ :

- ROS packages ได้ถูกติดตั้งไว้แล้วด้วย package managers provide setup (ตัวจัดการแพคเกจ) ในรูปของไฟล์ .*sh
- Rosbuild workspaces เป็นไฟล์นามสกุล .*sh ซึ่งใช้ไฟล์ setup.*sh เป็น tools เช่น rosws
- ไฟล์ Setup. *sh ที่ถูกสร้างไว้เป็นแบบ by-product building หรือ by- product installing catkin packages

ถ้าหากผู้เรียนได้ติดตั้ง ROS ด้วยคำสั่ง apt บนระบบปฏิบัติการ Ubuntu ดังนั้นแสดงว่าผู้เรียนจะสามารถตั้งค่า setup.*sh ไฟล์ ไว้ใน `'/opt/ros/<distro>/'` ได้ และผู้เรียนยังสามารถทำการ source ไฟล์ดังกล่าวได้ด้วยคำสั่งด้านล่างนี้

```
$ source /opt/ros/<distro>/setup.bash
```

*โดย `<distro>` ให้ใส่เป็น ROS distribution แทน

ถ้าผู้เรียนติดตั้ง ROS เป็นแบบ kinetic การ source จะเป็นดังคำสั่งด้านล่างนี้

```
$ source /opt/ros/kinetic/setup.bash
```

***** ผู้เรียนจะต้อง Run command แบบนี้ทุกครั้ง !!! ที่มีการเรียก shell หรือ terminal ใหม่ขึ้นมา หากจะต้องมีการใช้ คำสั่งต่างๆใน ROS command ในบรรทัดต่อไป นอกจากว่าผู้เรียนจะ add คำสั่งตามบรรทัดด้านบนไว้ที่ .bashrc ของผู้เรียนเอง

ขั้นตอนวิธีที่กล่าวมานี้จะทำให้ผู้เรียนสามารถที่จะ ติดตั้ง ROS distributions หลายแบบ (เช่น indigo และ kinetic) บนคอมพิวเตอร์ของตนได้ โดยสลับการเข้าใช้งานได้ในแต่ละแบบ

หากมีการใช้ platform อื่น ผู้เรียนจะพบว่าไฟล์ setup.*sh จะอยู่ที่เดียวกับที่ที่ ROS ได้ถูกติดตั้งเอาไว้

3. การสร้างพื้นที่แห่งการทำงานให้กับ ROS (Create a ROS Workspace)

โดยเลือกใช้ catkin

3.1 ทำการ create และ build catkin workspace

```
4. $ mkdir -p ~/catkin_ws/src
5. $ cd ~/catkin_ws/
6. $ catkin_make
```

คำสั่ง catkin_make เป็นเครื่องมือที่ทำให้การทำงานบน catkin workspaces สะดวกขึ้น

การใช้งานคำสั่งนี้ตั้งแต่ครั้งแรกของการใช้งานบน workspace ของผู้เรียนนั้น จะทำให้เกิดการสร้างลิ้งค์ CMakeLists.txt ในโฟลเดอร์ src ขึ้น

Python 3 users in ROS Melodic and earlier: note, if you are building ROS from source to achieve Python 3 compatibility, and have setup your system appropriately (ie: have the Python 3 versions of all the required ROS Python packages installed, such as `catkin`) the first catkin_make command in a clean catkin workspace must be:

```
$ catkin_make -DPYTHON_EXECUTABLE=/usr/bin/python3
```

This will configure catkin_make with Python 3. You may then proceed to use just `catkin_make` for subsequent builds.

นอกจากนี้ ถ้ามองกลับไปที่ directory ที่ผ่านมา จะเห็นว่าตอนนี้เรามีโฟลเดอร์ที่ชื่อ buid และ devel โดยภายในโฟลเดอร์ devel จะเห็นไฟล์ setup.*sh อีกหลายไฟล์อยู่ภายใน

การ sourc ไฟล์ข้างต้นจะทำให้เกิดการทำงานโดยที่วางทับ workspace ด้านบนของ environment นั้นไปเลย

จากนั้นก่อนจะเริ่มรันคำสั่งต่อต้องทำการ source ไฟล์ setup.*sh เสียก่อน

```
$ source devel/setup.bash
```

เพื่อยืนยันให้แน่ใจว่าตอนนี้ workspace ของเราได้ถูก คำสั่ง setup ข้างต้นซ้อนทับไปแล้ว โดยการเช็ค ตัวแปรชื่อ ROS_PACKAGE_PATH ใน environment ว่ามีอยู่ใน directory ที่เราอยู่ในตอนนี้ไหมด้วยคำสั่งด้านล่าง

```
$ echo $ROS_PACKAGE_PATH
/home/youruser/catkin_ws/src:/opt/ros/kinetic/share
```

Navigating the ROS Filesystem

การค้นหาไฟล์ต่างๆของระบบ ROS

คำอธิบายเพิ่มเติม : ใน tutorials นี้ เน้นอธิบายไปที่คอนเซปต์ ของระบบการจัดการไฟล์ต่างๆของ ROS รวมถึงอธิบายการใช้เครื่องมือของ commandline เช่น roscd , rosls และ rospack

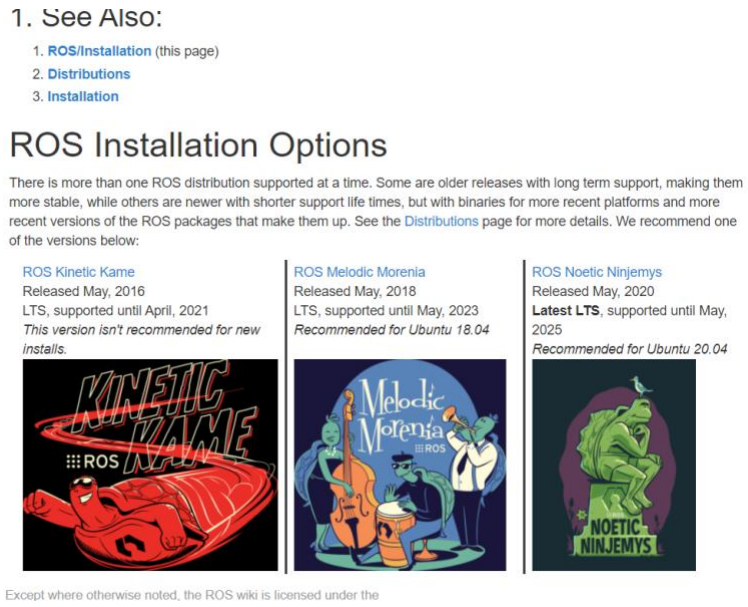
ในที่นี้เนื่องจากเรายังเป็นมือใหม่สำหรับ ROS จะทำการเลือก catkin แทน rosbuild ในการทำความเข้าใจ

1. ข้อกำหนดเบื้องต้น

สำหรับ tutorial นี้เราจะติดตั้ง package ใน ros-tutorials โดยใช้คำสั่งด้านล่าง

โดยเปลี่ยน '<distro>' เป็น indigo , kinetic,lunar และอื่นๆ ตามที่เราได้ติดตั้งไว้

```
$ sudo apt-get install ros-<distro>-ros-tutorials
```



1. See Also:
   1. **ROS/Installation** (this page)
   2. **Distributions**
   3. **Installation**

ROS Installation Options

There is more than one ROS distribution supported at a time. Some are older releases with long term support, making them more stable, while others are newer with shorter support life times, but with binaries for more recent platforms and more recent versions of the ROS packages that make them up. See the Distributions page for more details. We recommend one of the versions below:

ROS Kinetic Kame	ROS Melodic Morenia	ROS Noetic Ninjemys
Released May, 2016	Released May, 2018	Released May, 2020
LTS, supported until April, 2021	LTS, supported until May, 2023	**Latest LTS**, supported until May, 2025
This version isn't recommended for new installs.	*Recommended for Ubuntu 18.04*	*Recommended for Ubuntu 20.04*

2. ภาพรวมของคอนเซปต์ ระบบไฟล์ ROS

   Packages : Packages เป็นซอฟแวร์เพื่อจัดระเบียบการประมวลผลโค้ดของ ROS หน่วยหนึ่ง

   Manifests (package.xml): เป็น package ชนิดหนึ่ง ทำหน้าที่ในการระบุตำแหน่งระหว่าง package และ เก็บข้อมูล package เช่น version, maintainer, license และอื่นๆ

3. เครื่องมือในการจัดการระบบไฟล์

   เนื่องจาก package ต่างๆของ ROS เต็มไปด้วยโค้ดคำสั่ง การใช้เครื่องมือในการค้นหาของ command-line เช่น ls , cd ในการค้นหาเพียงอย่างเดียวนั้นไม่สะดวกเท่าที่ควร ทาง ROS จึงสร้างเครื่องมือใหม่มาให้เราด้วย

   3.1 การใช้ rospack

   Rospack ใช้เพื่อหาข้อมูลของ package ต่างๆ โดยในที่นี้ เราจะใช้เพียงแค่ option ที่มีชื่อว่า `find` ในการค้นหา path ของไฟล์ที่เราต้องการทราบผ่านคำสั่งด้านล่างนี้

```
$ rospack find [package_name]
```

เช่น

```
$ rospack find roscpp
```

จะแสดงผลออกมาเป็น

```
YOUR_INSTALL_PATH/share/roscpp
```

โดยถ้าเราได้มีการติดตั้ง ROS Kinetic จากคำสั่ง `apt` บน Ubuntu Linux จะเห็นการแสดงผลเป็น

```
/opt/ros/kinetic/share/roscpp
```

3.2 การใช้ roscd

roscd เป็นส่วนหนึ่งของ rosbash ที่ทำให้เราสามารถเข้าไปเปลี่ยน directory ของ package ต่างๆ หรือ stack ได้โดยตรงจากชุดคำสั่งด้านล่าง

```
$ roscd <package-or-stack>[/subdir]
```

หากต้องการตรวจสอบว่าเราเปลี่ยน directory ของ roscpp package สำเร็จแล้วหรือยัง ให้ Run คำสั่งด้านล่าง

```
$ roscd roscpp
```

ทีนี้ลองสั่งแสดงผล directory ที่กำลังทำงานโดยใช้คำสั่งจาก Unix command

```
$ pwd
```

เราจะเห็น

```
YOUR_INSTALL_PATH/share/roscpp
```

คุณจะเห็นว่า path `YOUR_INSTALL_PATH/share/roscpp` เป็น path เดียวกันกับที่ใช้ชุดเครื่องมือคำสั่งการจัดการไฟล์ `rospack find` จากตัวอย่างก่อนหน้า

หมายเหตุ : roscd นั้นก็เหมือนกัน ROS tools ตัวอื่น มันจะสามารถหาได้เฉพาะ ROS package ที่อยู่ใน directories listed (ไดเรกทอรี่ที่ปรากฏ) ใน ROS_PACKAGE_PATH โดยการเช็คว่ามีอะไรใน ROS_PACKAGE_PATH ของคุณ จากการพิมโค้ดคำสั่งด้านล่าง

```
$ echo $ROS_PACKAGE_PATH
```

ตอนนี้ ROS_PACKAGE_PATH จะโชว์ directories ต่างๆ ที่คุณได้เก็บข้อมูลของ ROS packages ต่างๆไว้ออกมาโดยแต่ละ path จะคั่นด้วยเครื่องหมาย colons ‘ : ’

โดยทั่วไป ROS_PACKAGE_PATH จะแสดงผลออกมาตามด้านล่างนี้

```
/opt/ros/kinetic/base/install/share
```

3.2.1 Subdirectories

roscd สามารถใช้ย้าย directory ไปยัง subdirectory ของ package หรือ stack ได้ด้วยเช่นกัน โดยทดลองพิมพ์

```
$ roscd roscpp/cmake
$ pwd
```

เราจะเห็น

```
YOUR_INSTALL_PATH/share/roscpp/cmake
```

3.3 roscd log

`roscd log` เป็นคำสั่งพาไปสู่ที่เก็บ log files ของ ROS โดยหากยังไม่มีการ run โปรแกรมใดๆเลย ใน ROS แล้ว run คำสั่งนี้ จะทำให้เกิด error : it does not yet exist

หากเราได้ทำการ run คำสั่งบางอย่างบน ROS เรียบร้อยแล้วให้ทดลองพิมพ์ตามคำสั่งด้านล่างนี้

```
$ roscd log
```

3.4 การใช้ rosls

rosls เป็นส่วนหนึ่งของ rosbash ที่สามารถทำให้เราเข้าไปสู่ package ได้ด้วยชื่อเลย โดยไม่จำเป็นต้องเข้าถึงด้วย path

```
$ rosls <package-or-stack>[/subdir]
```

เช่น

```
$ rosls roscpp_tutorials
```

จะแสดงผลเป็น

```
cmake launch package.xml  srv
```

3.5 Tab Completion

การเข้าถึงไฟล์นั้นหากไฟล์มีชื่อที่ยาวเกินไปการพิมพ์ชื่อทั้งหมดอาจไม่ใช่วิธีที่สะดวกนัก เช่น roscpp_tutorials โชคดีที่ ROS ยังมีเครื่องมือที่ใช้ในการพิมพ์ชื่อไฟล์ยาวๆมาให้เราใช้กัน โดยการกด TAB ตามด้านล่างนี้

```
$ roscd roscpp_tut<<< กด TAB >>>
```

หลังจากกด ปุ่ม TAB บนแป้นพิมพ์ command line จะแสดงผลดังนี้

```
$ roscd roscpp_tutorials/
```

หากแสดงผลบรรทัดด้านบนได้เลย เพราะว่า `roscpp_tutorials` ปัจุบันมีเพียงไฟล์เดียว ใน ROS package ที่ขึ้นต้นด้วย roscpp_tut.

ทีนี้ลองพิมพ์

```
$ roscd tur <<< กด TAB >>>
```

หลังจากที่เราได้กด TAB การแสดงผลที่ command line จะโชว์ชื่อไฟล์ที่คล้ายคำด้านบนทั้งหมดออกมา

```
$ roscd turtle
```

อย่างไรก็ตามในกรณีนี้มีหลาย package ที่ขึ้นต้นด้วยตัวอักษร turtle. เราจะมาทอลองพิมพ์ TAB อีกครั้ง โดยคราวนี้การแสดงผลจะแสดงชื่อที่ ขึ้นต้นด้วย turtle ทั้งหมดที่มีอยู่ในROS package

```
turtle_actionlib/  turtlesim/         turtle_tf/
```

นอกจากนี้บน command line เราอาจจะลองพิมพ์

```
$ roscd turtle
```

จากนั้นเพิ่มตัว `s` ข้างหลังคำว่า `turtle` จากนั้นกด TAB

```
$ roscd turtles<<< กด TAB >>>
```

จะเห็นว่ามีเพียง package เดียวที่โชว์ขึ้นมาที่ขึ้นต้นด้วย `turtles`

```
$ roscd turtlesim/
```

หรือถ้าหากเราต้องการที่จะเห็น package ที่ติดตั้งสำเร็จในตอนนี้ทั้งหมด ให้ทดลองทำตามบรรทัดด้านล่าง

```
$ rosls <<< กดปุ่ม TAB สองครั้ง>>>
```

4. Review

   ตอนนี้เราอาจจะเริ่มสังเกตุเห็นรูปแบบคำสั่งบางอย่างของ ROS tools ได้แล้ว

   rospack = ros + pack(age)

   roscd = ros + cd

   rosls = ros + ls

   ซึ่งการกำหนดชื่อในลักษณะนี้มีอยู่แทบจะทั้งหมดใน ROS tools

Creating a ROS Package
การ Create package ของ ROS

คำอธิบายเพิ่มเติม : ใน tutorials นี้จะอธิบายถึงการใช้ roscreat-pkg หรือ catkin ในการ create package ใหม่ และ ใช้ rospack ในการแสดงที่อยู่ของ package ต่างๆ

1. catkin Package ประกอบด้วยอะไรบ้าง

- ใน package จะต้องมีไฟล์ catkin compliant package.xml
  โดย package.xml นี้จะให้ข้อมูลเกี่ยวกับ package นั้นๆ
- ใน package จะมีไฟล์ CMakeLists.txt which uses catkin
  โดยถ้า catkin metapackage นั้นมีไฟล์ CMakeLists.txt ที่สำเร็จรูปแล้ว
- ในแต่ละ package จะต้องมีโฟลเดอร์เป็นของตนเอง
  ซึ่งนั้นหมายความว่า จะไม่มี package ที่ซ้อนกัน หรือ หลายๆ package ที่ใช้ directory ที่เหมือนกัน

package โดยพื้นฐานจะต้องมีลักษณะเหมือนด้านล่างนี้

```
my_package/
  CMakeLists.txt
  package.xml
```

2. Package ใน catkin Workspace

   จากวิธีการที่ได้เคยแนะนำไป กับการทำงานของ catkin package นั้นเราใช้ catkin workspace ในการทำงาน แต่ เราเองก็สามารถสร้าง catkin workspace แบบ stand alone ได้เช่นเดียวกัน โดยมีโครงสร้างบางส่วนเป็นดังนี้

```
workspace_folder/        -- WORKSPACE
  src/                   -- SOURCE SPACE
    CMakeLists.txt       -- 'Toplevel' CMake file, provided by catkin
    package_1/
      CMakeLists.txt     -- CMakeLists.txt file for package_1
      package.xml        -- Package manifest for package_1
    ...
    package_n/
      CMakeLists.txt     -- CMakeLists.txt file for package_n
      package.xml        -- Package manifest for package_n
```

3. การ Create catkin Package

   ในคู่มือนี้จะสาธิตวิธีการใช้ catkin_create_pkg ในการสร้าง(create) catkin package ใหม่ และ

   ขั้นตอนต่อไปหากเราสร้าง(create) Package สำเร็จแล้ว

   ขั้นตอนแรก : เปลี่ยนที่อยู่ของ catkin_workspace ที่เราได้สร้าง(create)ไว้แล้ว

   (http://wiki.ros.org/catkin/Tutorials/create_a_workspace)

   ไปไว้ที่ source space directory

```
# You should have created this in the Creating a Workspace Tutorial
$ cd ~/catkin_ws/src
```

   หลังจากนั้นเราจะใช้ `catkin_create_pkg` ในการสร้าง(create) Package ใหม่ขึ้นมา โดยตั้งชื่อว่า beginner_tutorials ซึ่งประกอบไปด้วย std_msgs, roscpp, และ rospy

```
$ catkin_create_pkg beginner_tutorials std_msgs rospy roscpp
```

ตอนนี้เราจะได้โฟลเดอร์ชื่อ beginner_tutorials มาแล้วซึ่งประกอบไปด้วย package.xml และ CMakeLists.txt อยู่ภายใน โดยข้อมูลบางส่วนอาจจะยังไม่สมบูรณ์ขึ้นอยู่กับข้อมูลที่เราได้ให้ไว้ใน `catkin_create_pkg`

ในส่วนของ `catkin_create_pkg` เองนั้นก็ต้องการข้อมูลของ `package_name` และ ทางเลือกในการตั้งชื่ออื่นๆ (A list of dependencies) เรียงเป็นลำดับ จากเรา

```
# ส่วนนี้เป็นเพียงแค่ตัวอย่างไม่สามารถ Run ได้
# catkin_create_pkg <package_name> [depend1] [depend2] [depend3]
```

4. การสร้าง (Build) catkin workspace และ การ source ไฟล์ setup

   ขั้นแรกเริ่มด้วยการสร้าง (Build) Packages ใน catkin workpace

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

หลังจากที่สร้าง (build) workspace เสร็จแล้วภายในจะถูกสร้าง(create)ไว้ด้วยโครงสร้างที่คล้ายกับโครงสร้างในโฟล์เดอร์ย่อยที่ชื่อ `devel` ซึ่งปกติเราสามารถเข้าดูโครงสร้างได้จาก `/opt/ros/$ROSDISTRO_NAME`

ในการเพิ่ม workspace ใน ROS environment ของเรา เราจำเป็นต้องทำการ source ไฟล์ setup ที่เราถูกสร้างขึ้นมาเสียก่อน

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

5. package dependencies

5.1 First-order dependencies

เมื่อเราได้มีการเรียกใช้ catkin_create_pkg มาก่อนแล้ว จะมี package dependencies จำนวนหนึ่งปรากฏขึ้นมา โดยเราสามารถเรียกดู First-order dependency นี้ได้ โดยทำการ Run คำสั่งด้านล่างจาก rospack tool

```
$ rospack depends1 beginner_tutorials
```

```
roscpp
rospy
std_msgs
```

จากผลลัพธ์ด้านบน rospack ได้ทำการเรียง dependencies ที่ทำหน้าที่เหมือนกันคือถูกใช้เป็น argument ขณะที่ `catkin_create_pkg` กำลังทำงาน โดย dependencies ของ package เหล่านี้จะถูกเก็บไว้ในไฟล์ package.xml อีกที

```
$ 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

ในหลายกรณี dependency นั้นมักจะมี dependency ของตัวเองด้วยอีกที ดังเช่น rospy นั้น ก็จะมี dependency ของตัวเองด้วยดังนี้

```
$ rospack depends1 rospy
```

```
genpy
roscpp
rosgraph
rosgraph_msgs
roslib
std_msgs
```

Package นั้นบางทีมี indirect dependency จำนวนไม่มากแต่จำเป็นต้องเรียกใช้ซ้อนกัน โชคดีที่เราสามารถใช้ rospack ในการเรียกซ้ำ หรือ เรียกใช้งาน dependency ตัวเดียวกันพร้อมกันได้

```
$ rospack depends beginner_tutorials
cpp_common
rostime
roscpp_traits
roscpp_serialization
catkin
genmsg
genpy
message_runtime
gencpp
geneus
gennodejs
genlisp
message_generation
rosbuild
rosconsole
std_msgs
rosgraph_msgs
xmlrpcpp
roscpp
rosgraph
ros_environment
rospack
roslib
rospy
```

6. การกำหนดค่า Package ด้วยตนเอง

6.1 การกำหนดค่า package.xml

package.xml ที่ถูกสร้างขึ้นมานั้นควรจะอยู่ใน package ใหม่ของเรา

6.1.1 description tag

เริ่มจากการตั้งค่า description tag

```
<description>The beginner_tutorials package</description>
```

เราสามารถเปลี่ยน description เป็นอะไรก็ได้ เพียงแต่ว่า คำแรก ของ description ควรสั้นในขณะที่ความหมายยังครอบคลุมหน้าที่ของแพคเกจนั้นๆด้วย แต่ถ้าคำเดียวอธิบายได้ไม่หมด เราอาจใช้วิธีการแยกคำในการตั้งชื่อได้เช่นกัน

6.1.2 maintainer tags

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

สิ่งนี้เป็นสิ่งที่จำเป็นสำหรับ package.xml เนื่องจากทำให้เราสามารถทราบช่องทางการติดต่อองกรณ์หรือบุคคล ที่เป็นผู้พัฒนา Package ที่เราใช้ได้ อย่างน้อยควรมี 1 คน หรือ 1ช่องทางการติดต่อ ซึ่งชื่อของผู้ที่พัฒนาจะอยู่ในส่วน body ของ tag รวมถึงข้อมูลการติดต่อก็จะอยู่ด้วยเช่นกัน

```
<maintainer email="you@yourdomain.tld">Your Name</maintainer>
```

6.1.3 licenses tags

```
  <!-- 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>TODO</license>
```

เราควรเลือก license และใส่ไว้ในนี้ เช่น license ที่เป็น open source อย่าง BSD, MIT, Boost Software License, GPLv2, GPLv3, LGPLv2.1, and LGPLv3 โดยในที่นี้จะขอยก BSD licenses มาประกอบการอธิบาย เนื่องจากเป็นส่วนหลักๆ ของ ROS

```
<license>BSD</license>
```

6.1.4 dependencies tag

โดย dependencies จะถูกแบ่งเป็น

`build_depend`, `buildtool_depend`, `exec_depend` และ `test_depend` จากหัวข้อที่ที่ผ่านๆมา หาก `std_msgs`, `roscpp` และ `rospy` เป็น arguments สำหรับ catkin_create_pkg จะมี dependencies รูปแบบดังด้านล่างนี้

```
<!-- The *_depend tags are used to specify dependencies -->
<!-- Dependencies can be catkin packages or system dependencies -->
<!-- Examples: -->
<!-- Use build_depend for packages you need at compile time: -->
<!--   <build_depend>genmsg</build_depend> -->
<!-- Use buildtool_depend for build tool packages: -->
<!--   <buildtool_depend>catkin</buildtool_depend> -->
<!-- Use exec_depend for packages you need at runtime: -->
<!--   <exec_depend>python-yaml</exec_depend> -->
<!-- Use test_depend for packages you need only for testing: -->
<!--   <test_depend>gtest</test_depend> -->
<buildtool_depend>catkin</buildtool_depend>
<build_depend>roscpp</build_depend>
<build_depend>rospy</build_depend>
<build_depend>std_msgs</build_depend>
```

ตอนนี้ dependencies ทั้งหมดของเราได้ถูกแทนด้วยคำว่า `build_depend`, `buildtool_depend` เรียบร้อยแล้ว และเรายังมีการเพิ่ม tag ที่มีชื่อว่า `exec_depend` เข้าไปข้างหน้า dependencies ที่มีลักษณะเฉพาะ (specified dependencies) เพื่อที่จะสามารถ build และ run ได้อยู่ตลอด

```
<buildtool_depend>catkin</buildtool_depend>

<build_depend>roscpp</build_depend>
<build_depend>rospy</build_depend>
<build_depend>std_msgs</build_depend>

<exec_depend>roscpp</exec_depend>
<exec_depend>rospy</exec_depend>
<exec_depend>std_msgs</exec_depend>
```

6.1.5 Final package.xml

As you can see the final package.xml, without comments and unused tags, is much more concise:

```
<?xml version="1.0"?>
<package format="2">
  <name>beginner_tutorials</name>
  <version>0.1.0</version>
  <description>The beginner_tutorials package</description>

  <maintainer email="you@yourdomain.tld">Your Name</maintainer>
  <license>BSD</license>
  <url type="website">http://wiki.ros.org/beginner_tutorials</url>
  <author email="you@yourdomain.tld">Jane Doe</author>

  <buildtool_depend>catkin</buildtool_depend>

  <build_depend>roscpp</build_depend>
  <build_depend>rospy</build_depend>
  <build_depend>std_msgs</build_depend>

  <exec_depend>roscpp</exec_depend>
  <exec_depend>rospy</exec_depend>
  <exec_depend>std_msgs</exec_depend>

</package>
```

6.2 การกำหนดค่า CMakeLists.txt

ตอนนี้ package.xml ที่ประกอบไปด้วยชุดข้อมูล meta (meta information) ได้ถูกปรับแต่งเพื่อให้สามารถเข้ากับ package ของเราได้เรียบร้อยแล้ว

ในส่วนของไฟล์ CMakeLists.txt ได้มีการสร้าง(create)ไว้แล้วด้วย catkin_create_pkg เราจะอธิบายในเอกสารชุดอื่นที่มีเนื้อหาเกี่ยวกับ การสร้าง (build) ROS code

Building a ROS Package

การ Build ROS Package

1. การ Build Package

   จากที่เราได้ทำการติดตั้ง system dependencies ของ package ทั้งหมดแล้ว เราจึงสามารถสร้าง (build) Package ใหม่ของเราได้

   โดยก่อนที่เราจะเริ่มสร้าง(build) Package ของเรา เราต้องทำการ source ไฟล์ environment setup เสียก่อน หากทำบนระบบปฏิบัติการ Ubuntu จะมีรูปแบบการแสดงผลดังด้านล่างนี้

```
# source /opt/ros/%YOUR_ROS_DISTRO%/setup.bash
$ source /opt/ros/kinetic/setup.bash             # For Kinetic for instance
```

   1.1 การใช้ catkin_make

   catkin_make เป็น tool ชนิดหนึ่งของ command line ที่อำนวยความสะดวกใน standard catkin workflow รวมถึงอาจจะมีการเรียก catkin_make ว่า cmake และ make ได้ใน standard CMake workflow ซึ่งมีรูปแบบการใช้ดังนี้

```
# In a catkin workspace
$ catkin_make [make_targets] [-DCMAKE_VARIABLES=...]
```

   สำหรับคนที่ยังไม่คุ้นชินกับมาตรฐาน CMake workflow สามารถศึกษาลักษณะการทำงานของ มาตรฐาน CMake workflow อย่างง่ายได้จากด้านล่าง

```
# In a CMake project
$ mkdir build
$ cd build
$ cmake ..
$ make
$ make install  # (optionally)
```

โดยกระบวนการนี้จะถูก Run เพื่อกระทำการใดๆก็ตามกับ CMake Project ในทางตรงข้าม ในส่วนของ catkin projects เองก็สามารถเกิดการ Build ไปพร้อมกันได้ใน workspace ของเรา

การเริ่มสร้าง catkin packages ตั้งแต่ยังไม่มีอะไรไปจนถึงทำให้มี catkin package จำนวนมากใน workspace สามารถทำตามได้ด้วยคำสั่งด้านล่างนี้

```
# In a catkin workspace
$ catkin_make
$ catkin_make install  # (optionally)
```

จากคำสั่งด้านบนจะทำให้เกิดการ build ในส่วนของ catkin project จำนวนมากโดยสามารถเข้าไปดูได้ที่โฟลเดอร์ src

แต่ถ้าหาก source code ของเรานั้นอยู่คนละที่กันโดย ในที่นี้จะยกตัวอย่างเป็น `my_src` เราจะต้องทำการเรียก catkin_make โดยใช้วิธีการตามด้านล่างนี้

```
# In a catkin workspace
$ catkin_make --source my_src
$ catkin_make install --source my_src  # (optionally)
```

1.2 การ Build Package ของเราเอง

หากว่าเรากำลังทำการ buid code ของเราอยู่ เราอาจจะต้องเข้าไปดู tutorial ในส่วนของ C++ หรือ Python ก่อนเนื่องจากเราอาจจะต้องมีการเข้าไปแก้ไข CMakeLists.txt

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

http://wiki.ros.org/ROS/Tutorials/WritingPublisherSubscriber%28python%29

ตอนนี้เราควรที่จะมี catkin workspace และ catkin package ใหม่ที่ชื่อว่า beginner_tutorials เรียบร้อยแล้วต่อเนื่องจาก tutorials ที่แล้ว (Creating a ROS Package)

แต่ถ้าหากเรายังไม่ได้อยู่ที่ catkin workspace เราจะต้องไปที่ catkin workspace เสียก่อนหรือ อาจจะหา catkin workspace ของเราจากโฟล์เดอร์ src ก็ได้

```
$ cd ~/catkin_ws/
$ ls src
```

```
beginner_tutorials/  CMakeLists.txt@
```

เราจะเห็นว่ามีโฟล์เดอร์ชื่อ beginner_tutorials ที่เราได้ทำการ Create ไว้แล้ว ด้วย catkin_create_pkg จาก tutorials ก่อนหน้า หลังจากที่เจอ catkin workspace เราก็สามารถทำการ buid ได้เลยโดยใช้ catkin_make

```
$ catkin_make
```

เราจะเห็น output ต่างๆมากมายจาก cmake และ make

```
Base path: /home/user/catkin_ws

Source space: /home/user/catkin_ws/src

Build space: /home/user/catkin_ws/build

Devel space: /home/user/catkin_ws/devel

Install space: /home/user/catkin_ws/install

####

#### Running command: "cmake /home/user/catkin_ws/src

-DCATKIN_DEVEL_PREFIX=/home/user/catkin_ws/devel

-DCMAKE_INSTALL_PREFIX=/home/user/catkin_ws/install" in "/home/user/cat
kin_ws/build"

####

-- The C compiler identification is GNU 4.2.1

-- The CXX compiler identification is Clang 4.0.0

-- Checking whether C compiler has -isysroot

-- Checking whether C compiler has -isysroot - yes

-- Checking whether C compiler supports OSX deployment target flag

-- Checking whether C compiler supports OSX deployment target flag - ye
s

-- Check for working C compiler: /usr/bin/gcc

-- Check for working C compiler: /usr/bin/gcc -- works

-- Detecting C compiler ABI info

-- Detecting C compiler ABI info - done

-- Check for working CXX compiler: /usr/bin/c++

-- Check for working CXX compiler: /usr/bin/c++ -- works

-- Detecting CXX compiler ABI info

-- Detecting CXX compiler ABI info - done

-- Using CATKIN_DEVEL_PREFIX: /tmp/catkin_ws/devel

-- Using CMAKE_PREFIX_PATH: /opt/ros/kinetic

-- This workspace overlays: /opt/ros/kinetic

-- Found PythonInterp: /usr/bin/python (found version "2.7.1")

-- Found PY_em: /usr/lib/python2.7/dist-packages/em.pyc

-- Found gtest: gtests will be built

-- catkin 0.5.51
```

```
-- BUILD_SHARED_LIBS is on
-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
-- ~~  traversing packages in topological order:
-- ~~  - beginner_tutorials
-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
-- +++ add_subdirectory(beginner_tutorials)
-- Configuring done
-- Generating done
-- Build files have been written to: /home/user/catkin_ws/build
####
#### Running command: "make -j4" in "/home/user/catkin_ws/build"
####
```

ให้จำไว้เสมอว่า catkin_make จะโชว์ path ต่างๆที่มีการใช้ ‘spaces’ ออกมาก่อนในช่วงแรก โดยความหมายของ ‘spaces’ คืออะไรเราสามารถศึกษาเพิ่มเติมได้จาก http://ros.org/reps/rep-0128.html และข้อมูลเพิ่มเติมเกี่ยวกับ catkin workspaces ได้จาก http://wiki.ros.org/catkin/workspaces มีสิ่งที่น่าสังเกตุอย่างหนึ่งว่าค่า default ต่างๆในหลายโฟลเดอร์ได้ถูก Create ไว้แล้วใน catkin workspace ของเรา สามารถดูได้จากการพิมพ์ ls

```
$ ls
```

```
build
devel
src
```

โดยโฟล์เดอร์ buid เป็น โฟล์เดอร์ที่เป็นตำแหน่ง default ของ build space และ เป็นที่ที่ cmake กับ make ถูกเรียกใช้เพื่อ configure และ build ให้กับ package ของเรา ส่วนโฟล์เดอร์ devel เป็นโฟล์เดอร์ที่เป็นตำแหน่ง default ขอว devel space ที่ซึ่งมี executables และ libraries อยู่ก่อที่เราจะทำการติดตั้ง packages