

Note: This tutorial assumes that you have completed the previous tutorials: Installing and Configuring Your ROS Environment (/ROS/Tutorials/InstallingandConfiguringROSEnvironment).

💡 Please ask about problems and questions regarding this tutorial on answers.ros.org (http://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.

Navigating the ROS Filesystem

Description: This tutorial introduces ROS filesystem concepts, and covers using the roscd, rosls, and rospack (/rospack) commandline tools.

Tutorial Level: BEGINNER

Next Tutorial: Creating a ROS package (/ROS/Tutorials/CreatingPackage)

Select "**catkin**" or "**roswuild**" just below. The new build system for ROS is "catkin", while "roswuild" is the old ROS build system which was replaced by catkin. If you are new to ROS, choose **catkin**. Here is some more info. about each:

- 1. catkin (/catkin)
- 2. catkin/conceptual_overview (/catkin/conceptual_overview)
- 3. catkin_or_roswuild (/catkin_or_roswuild)
- 4. roswuild (/roswuild)

catkin roswuild

Contents

- 1. Prerequisites
- 2. Quick Overview of Filesystem Concepts
- 3. Filesystem Tools
 - 1. Using rospack
 - 2. Using roscd
 - 1. Subdirectories
 - 3. roscd log
 - 4. Using rosls
 - 5. Tab Completion
- 4. Review

1. Prerequisites

For this tutorial we will inspect a package in ros-tutorials, please install it using

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

Replace '<distro>' (including the '<>') with the name of your ROS distribution (/Distributions) (e.g. indigo, kinetic, lunar etc.)

2. Quick Overview of Filesystem Concepts

- **Packages:** Packages are the software organization unit of ROS code. Each package can contain libraries, executables, scripts, or other artifacts.
- **Manifests (package.xml (/catkin/package.xml)):** A manifest is a description of a *package*. It serves to define dependencies between *packages* and to capture meta information about the *package* like version, maintainer, license, etc...

Hide () Note about stacks ()

Note: roswuild (/roswuild) users might be wondering where stacks went. The concept of stacks was removed with catkin to simplify the growing code base and to support better distribution of packages. In catkin (/catkin) you can define metapackages (/catkin/migrating_from_roswuild#Metapackages_and_the_Elimination_of_Stacks) to collect similar packages and multiple packages can reside in a single VCS repository. Those two features replace the functionality of stacks.

3. Filesystem Tools

Code is spread across many ROS packages. Navigating with command-line tools such as `ls` and `cd` can be very tedious which is why ROS provides tools to help you.

3.1 Using rospack

rospack (/rospack) allows you to get information about packages. In this tutorial, we are only going to cover the `find` option, which returns the path to package.

Usage:

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

Example:

```
$ rospack find roscpp
```

would return:

```
YOUR_INSTALL_PATH/share/roscpp
```

If you installed ROS Kinetic from apt on Ubuntu Linux you would see exactly:

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

3.2 Using roscd

roscd (/roscd) is part of the rosbash (/roscd) suite. It allows you to change directory ([cd](http://ss64.com/bash/cd.html) (http://ss64.com/bash/cd.html)) directly to a package or a stack.

Usage:

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

To verify that we have changed to the roscpp package directory, run this example:

```
$ roscd roscpp
```

Now let's print the working directory using the Unix command [pwd](http://ss64.com/bash/pwd.html) (http://ss64.com/bash/pwd.html):

```
$ pwd
```

You should see:

```
YOUR_INSTALL_PATH/share/roscpp
```

You can see that YOUR_INSTALL_PATH/share/roscpp is the same path that rospack find gave in the previous example.

Note that roscd (/roscd), like other ROS tools, will *only* find ROS packages that are within the directories listed in your ROS_PACKAGE_PATH (/ROS/EnvironmentVariables#ROS_PACKAGE_PATH). To see what is in your ROS_PACKAGE_PATH (/ROS/EnvironmentVariables#ROS_PACKAGE_PATH), type:

```
$ echo $ROS_PACKAGE_PATH
```

Your ROS_PACKAGE_PATH (/ROS/EnvironmentVariables#ROS_PACKAGE_PATH) should contain a list of directories where you have ROS packages separated by colons. A typical ROS_PACKAGE_PATH (/ROS/EnvironmentVariables#ROS_PACKAGE_PATH) might look like this:

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

Similarly to other environment paths, you can add additional directories to your ROS_PACKAGE_PATH (/ROS/EnvironmentVariables#ROS_PACKAGE_PATH), with each path separated by a colon ':'.

3.2.1 Subdirectories

roscd (/roscd) can also move to a subdirectory of a package or stack.

Try:

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

You should see:

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

3.3 roscd log

roscd log will take you to the folder where ROS stores log files. Note that if you have not run any ROS programs yet, this will yield an error saying that it does not yet exist.

If you have run some ROS program before, try:

```
$ roscd log
```

3.4 Using rosls

rosls (/rosls) is part of the rosbash (/roscd) suite. It allows you to [ls](http://ss64.com/bash/lis.html) (http://ss64.com/bash/lis.html) directly in a package by name rather than by absolute path.

Usage:

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


Example:

```
$ rosls roscpp_tutorials
```

would return:

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

3.5 Tab Completion

It can get tedious to type out an entire package name. In the previous example, `roscpp_tutorials` is a fairly long name. Luckily, some ROS tools support  TAB completion (http://en.wikipedia.org/wiki/Command_line_completion).

Start by typing:

```
$ roscd roscpp_tut<<< now push the TAB key >>>
```

After pushing the **TAB** key, the command line should fill out the rest:

```
$ roscd roscpp_tutorials/
```

This works because `roscpp_tutorials` is currently the only ROS package that starts with `roscpp_tut`.

Now try typing:

```
$ roscd tur<<< now push the TAB key >>>
```

After pushing the **TAB** key, the command line should fill out as much as possible:

```
$ roscd turtle
```

However, in this case there are multiple packages that begin with `turtle`. Try typing **TAB** another time. This should display all the ROS packages that begin with `turtle`:

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

On the command line you should still have:

```
$ roscd turtle
```

Now type an `s` after `turtle` and then push **TAB**:

```
$ roscd turtles<<< now push the TAB key >>>
```

Since there is only one package that starts with `turtles`, you should see:

```
$ roscd turtlesim/
```

If you want to see a list of all currently installed packages, you can use tab completion for that as well:

```
$ rosls <<< now push the TAB key twice >>>
```

4. Review

You may have noticed a pattern with the naming of the ROS tools:

- `rospack` = `ros` + `pack`(age)
- `roscd` = `ros` + `cd`
- `rosls` = `ros` + `ls`

This naming pattern holds for many of the ROS tools.

Now that you can get around in ROS, let's  create a package (<http://wiki.ros.org/ROS/Tutorials/CreatingPackage>).

Except where otherwise noted, the ROS wiki is licensed under the Creative Commons Attribution 3.0 (<http://creativecommons.org/licenses/by/3.0/>)

Wiki: ROS/Tutorials/NavigatingTheFilesystem (last edited 2020-09-21 17:34:03 by Himanshu (/Himanshu))

Brought to you by:  Open Robotics

(<https://www.openrobotics.org/>)