# ROS developer's guide

1. Source control
   - Git

=> Add to source control only the minimal set of manually written source & build files that are necessary to build your package.

- Dont add machine generated files such as:
  - Objects (.o)
  - libraries (.so)
  - auto-generated configure script.
- Dont add large binary files:
  - Upload them to a web server and download them in your build file.

*

=> Commit your code early & often.

=> Try to keep commits focused on a particular change instead of lumping multiple changes together.

=> Give an informative message with each commit.

=> Dont break the build.

2. Bug tracking

⇒ For package hosted on GitHub this is commonly the issue tracker of the repository.

⇒ The maintainer will assign milestone to each reported issue.

⇒ To raise an issue:

(1) Be as descriptive as possible.

→ Include instructions for reproducing the bug.

3. Code layout

⇒ Packages can be collected in a single repository.

⇒ For GitHub it is recommended to create a README.md at the root of your repository to explain to users what to find in the repository.

⇒ It is recommended to link to the Package documentation on the ROS wiki for the contained package.

4. Packaging

⇒ The ROS package and build system relies on package.xml files.

- Every package must have a package.xml file, located in the package's top directory.
- At a minimum, the package.xml must contain:
  - description
  - author
  - license

5. GUI toolkits

- Use rqt for any new GUI development.

{Qt-based GUI framework for ROS}

6. Building

⇒ The basic build tool is CMake.

- Every package that has a build step must have a CMakeList.txt file in the package's top directory.
- Packages that don't have build steps don't need any build files.

7. Licensing

~~⇒ We prefer permissive~~

⇒ The preferred license for the Project is the BSD license.

↳ Whenever possible, new code should be licensed under BSD.

⇒ The full text statements of all licenses used in the project should be placed in the LICENSES directory at the top of the repository.

⇒ Every source file should contain a commented License Summary at the top.

## (8) Copyright

⇒ Under the Berne Convention, the author of a work automatically holds copyright, with or without a formal statement to that effort.

↳ However, making copyright explicit is helpful in long-term project management

⇒ Each source file should contain a commented copyright line at the top

eg: Copyright 2008 Jim Bob

{This is usually directly above the license statement}

⇒ If you work for yourself, then you own the copyright.

⇒ If you work for someone else, then your employer owns the copyright.

---

⇒ The most popular open source license has an important aspect in common.

↳ The Open Source Initiative (OSI) has approved them.

formed in 1998

Applying a License to your open source project

↳ You need to add a LICENSE, LICENSE.txt or LICENSE.md in the root directory of your repository.

---

## ⑨ Debugging

⇒ Generally used debugging tools used in ROS:

- GDB
- Oprofile
- Valgrind

10) Testing

⇒ We use two level of testing

- Library ⇒ At the library level, we use standard unit-test frameworks.
  - → In C++, we use gtest
  - → In Python, we use unittest.
- Message ⇒ At the message level, we use rostest to set up a system of ROS nodes, run a test node, then tear down the system.

11) Documentation

⇒ All code should be documented according to QA Process. Including
- → All externally visible code-level API must be documented.
- → All externally visible ROS-level APIs (topic, services, Parameters) must be documented.

12) Releasing

⇒ The standard process for releasing code to the ROS community is described on the bloom documentation.

13)

14)

13) Standardization

⇒ Code should use ROS services, follow guidelines for their use:-

- Use rosout for printing message
- Use the ROS clock for time based routines.

14) Deprecation

⇒ As soon as there are users of your code, you have a responsibility not to pull the rug out from under them with sudden breaking changes.

- Instead use the process of deprecation, which means making a feature or component as being no longer supported, with a schedule for its removal.

⇒ Deprecation can happen at multiple levels:

1. API feature
   - Mark it as deprecated in the API documentation; with Doxygen.
2. Package
   - Mark it deprecated in Wiki documentation.

15) Large data files & Large test files

⇒ Large data files should be hosted in a public web hosting site.

↳ {anything over 1MB}

⇒ To download this file for building use the catkin_download_test_data.