



Published on *ROOT* (<http://root.cern.ch/drupal>)

[Home](#) > [Printer-friendly PDF](#) > [Printer-friendly PDF](#)

Installing ROOT from Source

[building](#) ^[1] [compiling](#) ^[2] [installing](#) ^[3] [source](#) ^[4]

Getting the Source

ROOT's source can be [downloaded for each of the releases](#) ^[5] and unpacked with

```
gzip -dc root_<version>.source.tar.gz | tar -xf -
```

Alternatively one can download ROOT's source from Subversion.

Subversion

Unlike many other projects, ROOT's trunk [is expected to always work](#) ^[6]. It is a snapshot of the current development; we appreciate feedback and people (you!) using it. The main advantages of using the trunk are:

- you get the most recent features
- you can easily benefit from bug fixes should you find one
- to fix or extend ROOT you can change ROOT's sources yourself and send the changes ([svn diff](#)) as feedback

The trunk can be obtained from our public [Subversion](#) ^[7] repository:

```
svn co http://root.cern.ch/svn/root/trunk [8] root
```

A specific tag can be obtained using:

```
svn co http://root.cern.ch/svn/root/tags/v5-26-00b [9] root-5.26.00b
```

Getting Ready to Build

To successfully build ROOT a number of prerequisite packages must be installed. Check the [prerequisites page](#) ^[10] for the list of packages needed for your platform.

You may want to compile features into ROOT, which depends on third party libraries. Make sure you meet all demands for additional features before trying to enable them (see below).

Otherwise, before proceeding, read the rest below.

Two build methods

ROOT can be build either using the classic `./configure;make` method or using [CMake](#) ^[11]. The advantage of the first method is that it works *out-of-the box* on all Unix like machines, while on Windows it requires the cygwin environment to be installed. The advantage of cmake is that it works on all platforms, and can make projects for the native IDE's (like Visual Studio on Windows, or Xcode on OSX).

Follow [these instructions](#) ^[12] to install ROOT using CMake, or read below on using the `./configure;make` method.

Choosing the installation method

There are two main methods of installing ROOT from source: location independent and location dependent. The former is

advised for a personal installation of ROOT, while the later for a system wide installation. Both ways are described below.

Location independent installation

You want to install in a generic directory, depending solely on environment variables PATH and LD_LIBRARY_PATH.

1. Get the sources of the latest ROOT (see above)
2. Type the build commands:

```
cd root
./configure --help
./configure [<arch>] [set arch appropriately if no proper default]
(g)make [or, make -j n for n core machines]
```

3. Add bin/ to PATH and lib/ to LD_LIBRARY_PATH. For the sh shell family do:

```
. bin/thisroot.sh
```

and for the csh shell family do:

```
source bin/thisroot.csh
```

4. Try running ROOT:

```
root
```

On MacOS X >= 10.5 this is the preferred installation method, as ROOT is build with an option that allows the library path to be derived from the location of the executable. This make the MacOS X built fully relocatable and independent of (DY)LD_LIBRARY_PATH.

To get rid of the intermediary build products do: `make clean` (and don't do: `make install`, which is a NOOP).

Fixed location installation

You want to install ROOT in a fixed location, not depending on LD_LIBRARY_PATH. Typically you want to do this when installing in a system location like `/usr/local/bin` or even `/usr/bin`. This allows you, and all other users on your system, to run ROOT without having to setup special PATH and LD_LIBRARY_PATH variables. This scheme also makes it easier to make ROOT distributions/installations that fit with an existing system, e.g. it should be trivial to install ROOT via a RedHat (.rpm) or Debian GNU/Linux (.deb) package. It's **strongly** recommended to enable version numbers in sonames (if possible) using this method (see `--enable-soversion` below). Also, using the `--enable-rpath` option to `./configure` you can set the load path on the ROOT libraries and applications. However, this is not recommended if you have some other way (like `/etc/ld.so.conf`) of making the dynamic loader aware of where you installed the ROOT libraries. Also note that the location of the `etc` directory needs to be explicitly set via the `--etcdir` option if the default of `/etc/root` is not desired.

1. Get the sources of latest ROOT (see above)
2. Now type the build commands:

```
./configure --help
./configure [<arch>] --prefix=/usr/local [set arch appropriately if no proper default]
(g)make [or, make -j n, for n core machines]
(g)make install [sudo or su to root if --prefix points to a system dir]
```

3. Update the ldconfig cache by doing:

```
ldconfig
```

4. Try running ROOT:

```
root
```

Installation Location

By default, the system will be installed location independent. In that case the whole package will live under the directory it was built in. This directory can be moved around and only PATH and LD_LIBRARY_PATH need to be updated. This is best done by just rerunning the `thisroot.[c]sh` script as mentioned above.

Using the fixed location mode, the default `--prefix` path is `/usr/local`, which will result in the ROOT files to be installed in `/usr/local/bin`, `/usr/local/lib/root`, etc. You can specify an installation prefix other than `/usr/local` by giving `./configure` the option `--prefix=<PATH>`. For more detailed location specifications see `./configure --help`.

Here's the complete list of `./configure` location options:

- `--prefix (/usr/local)`
Installation prefix. This will prefix any installation directory not explicitly specified.
- `--bindir (<prefix>/bin)`
This is where the ROOT applications (root, cint, rootcint, etc.) will be installed.
- `--libdir (<prefix>/lib/root)`
Library installation directory. All the class libraries of ROOT will be installed into this directory. You should make your dynamic linker aware of this directory. On Linux - and some other Un*xes - this directory can be added to `/etc/ld.so.conf` and `ldconfig` should be run afterward (you need to be root - the user - to do this). Please note, that this directory should probably not be something like `/usr/lib` or `/usr/local/lib`, since you'll most likely get a name clash with ROOT libraries and other libraries (e.g. `libMatrix.so`); rather use something like `/usr/local/lib/root`
- `--incdir (<prefix>/include/root)`
Header installation directory. All the header (declaration) files for the ROOT classes will be installed into this directory. This should be parallel to `libdir`, for consistency.
- `--etcdir (/etc/root)`
Configuration files installation directory. The system-wide `system.rootrc` and `root.mimes` will be installed into this directory.
- `--mandir (<prefix>/share/man/man1)`
Installation directory for the ROOT `man(1)` pages. This should be somewhere searched by `man(1)`. On most Un*x you can set the search path for `man(1)` via environment variable `MANPATH`. On some systems, a special configuration file `/etc/manpath.config` exist and should be used. See also `man(1)` - type `man man` in the prompt - on your system.
- `--datadir (<prefix>/share/root)`
Top-level data installation directory. Under this directory, various data files needed by ROOT will be installed, either in subdirectories, or directly.
- `--proofdir (<datadir>/proof)`
PROOF utilities directory. Various PROOF utility scripts as well as example configurations are installed into this directory.
- `--macrodir (<datadir>/macros)` Macro installation directory. Macros - properly called scripts - provided by the ROOT system are installed into this directory. ROOT - the application - will always search this directory for scripts.
- `--cintincdir (<datadir>/cint)`
CINT data directory. In this directory, the CINT runtime headers will be installed, both the standard headers, as well as pre-processed once.
- `--iconpath (<datadir>/icons)`
Icon installation directory. The icons used by the TBrowser and other classes will be installed into this directory. The user can specify additional directories in using the `"--with-sysicondir"` option.
- `--srcdir (<datadir>/src)`
Sources installation directory. Currently not used!
- `--docdir (<prefix>/doc/root)`
Documentation, like the LICENCE, README, etc. files will be installed into this directory.
- `--testdir (<docdir>/test)`
The test applications and libraries will be installed here.
- `--tutdir (<docdir>/tutorial)`
All the tutorials will be installed in this directory.

Makefile targets

The Makefile layout and supported make targets are explained in the file [README/BUILDSYSTEM](#) ^[13] available in the ROOT sources.

Installing optional plugin libraries

If you want to compile the ROOT optional plugin libraries to handle advanced math (GSL), Globus or SRP authentication, MySQL access, Castor access, event generator interfaces (Pythia6 and Pythia8), etc. you can either specify the options as environment variables or as `./configure` arguments. For example:

```
# Used during build of ROOT can be overridden in ./configure
export ROOTBUILD=debug          # see README/BUILDSYSTEM
export RFTIO=~/.shift/lib       # CERN's SHIFT library, must
                                # contain libshift.a

export GSL=~/.src/gsl-1.12
export FFTW3=~/.src/fftw-3.1.2
#export AFS=                    # must contain include/ lib/
#export MYSQL=                  # must contain include/ lib/
#export PYTHIA6=                 # must contain libPythia6
```

The ROOTBUILD environment variable is special and architecture dependent. To get an idea of which values it can take, take a look in the `config/Makefile.<arch>` corresponding to your architecture.

Using `configure` arguments

You can also specify additional features and external linking using command line options - or flags - to the `configure` script.

A special flag is `--build=<option list>` which correspond to the ROOTBUILD environment variable, as described above.

The flags consist of two classes:

1. `--enable-<feature>/--disable-<feature>` flags:

These flags enables/disables support for the corresponding feature `<feature>`. Specifying such a flag will make the `configure` script look for third-party libraries, headers, etc. at default locations.

If you find that the default locations does not work with some standard path e.g., on OS `<Foo>`, library `lib<Bar>`, is in general installed in path `<Baz>`, but "`configure`" doesn't look there, please file a [bug report](#) [14].

If, for some reason, you have installed some third-party library outside it's usual location, you can make `./configure` look for it using the corresponding `--with-<feature dir>` option (see below).

By default, all features are enabled, except for a few cases (noted below).

Here's a list of supported features:

- `shared`
Use shared 3rd party libraries if possible. Some platforms assume this by default, others do not support it at all. Default is true.
- `rpath`
Set library path on executables. This is only supported on some platforms. Default is false.
- `soversion`
Include the major version number of ROOT in the soname of the shared libraries. This means that you'll have an additional check on whether your applications are linked to the proper runtime libraries. It's strongly encouraged to use this option. This options is only supported on some platforms. Default is false.
- `table`
Build STAR contributed library, including new containers and extended 3D geometry descriptions. Default is false.
- `opengl`
OpenGL (3D rendering and visualisation) support. This is especially powerful if you have a 3D accelerated card and a proper OpenGL interface for your graphics system. This requires the third party libraries `libGL` and `libGLU` - or `libMesaGL` and `libMesaGLU`, as well header files for these libraries. For platforms with native OpenGL support (Windows), this is always enabled, even if the `--disable-opengl` flag is set. Default is true.
- `mysql`
Build a thin MySQL client for ROOT. This requires the third-party library `libmysqlclient` and header files. Note that there may be licensing issues regarding linking against `libmysqlclient`. Default is true.
- `pgsql`
Build a thin PostgreSQL client for ROOT. This requires the third-party library `libpq` and header files. Default is true.
- `sapdb`
Build a thin SapDB client for ROOT. This requires the third-party library `libsqlod` and header files. Default is true.
- `rfio`
Interface to SHIFT managed tape robots. This option requires you get the `libshift.a` from CERN. Please note, that this library is only available as a static library (archive), and will always be statically linked in. There may be licensing issues involved. Default is true.
- `pythia6`
Build thin wrapper for the Pythia Event Generator (version 6). This requires the third party library `libPythia6`. Default is true.
- `pythia8`
Build thin wrapper for the Pythia Event Generator (version 8). This requires the third party library `libpythia8`. Default is true.
- `soversion` Set version number in sonames. Default is false.

2. `--with-<feature dir>=<dir>` flags: If `./configure` isn't able to find some third-party library, header, or other file, corresponding to one of the above features, you can force it to look for these files in specific directories, using the `--with-` options.

Using a `--with-<feature dir>=<dir>` will make `./configure` look for whatever needed by `<feature>` in `<dir>`, and

will also enable the corresponding feature. Hence, you do not need to specify "`--enable-<feature>`" if you give an "`--with-<featuringdir>=<dir>`". The inverse - of course - isn't true.

Here is a list of "`--with-`" flags:

- `sys-iconpath`
Extra icon path for TBrowser, etc. This, again doesn't correspond to any feature, but provides you with way of getting the TBrowser and others to look for additional icons in some system-wide directory.
- `opengl-incdir`
Directory containing GL/gl.h (note the sub-directory), needed by feature `opengl`.
- `opengl-libdir`
Directory containing libGL.{so,a} and libGLU.{so,a} - alternatively libMesaGL.{so,a} and libMesaGLU.{so,a} - needed by feature `opengl`.
- `mysql-incdir`
Directory containing mysql.h, needed by feature `mysql`.
- `mysql-libdir`
Directory containing libmysqlclient.{so,a}, needed by feature `mysql`.
- `pythia6-libdir`
Directory containing libPythia6.{so,a}, needed by feature `pythia6`.
- `pythia8-incdir`
Directory containing Pythia8 includes, needed by feature `pythia8`.
- `pythia8-libdir`
Directory containing libpythia8.{so,a}, needed by feature `pythia8`.
- `shift-libdir`
Directory containing libshift.a, needed for feature `shify`.

As always, for the most up to date list of options see: `./configure --help`.

Information on third party software

OpenGL library

Most Linux distributions come with an OpenGL library (Mesa) pre-installed. The OpenGL libraries are already in XFree86 version 4 or above. On most Linux system the `mesa-libGL-devel-*` package must be installed. The Nvidia and ATI vendor drivers come often with their own optimized versions of OpenGL. On how to get the Mesa open source OpenGL see <http://www.mesa3d.org> ^[15].

CASTOR/SHIFT managed tape I/O

To build the library providing [CERN Castor/RPIO \(remote I/O\)](#) ^[16] support you need to get the Castor library from CERN. You can get pre-build libraries or the full source from [the Castor web site](#) ^[17].

MySQL client

To build the MySQL interface library you need to have `libmysqlclient.so` installed. On most Linux system this library comes with the `mysql-devel` package. For more see <http://www.mysql.com> ^[18].

Pythia Event Generators

To build the event generator interfaces for Pythia6, you first have to get the pythia libraries. You can import Pythia6 and the makefiles to build the Pythia6 shared library on several platforms from: <ftp://root.cern.ch/root/pythia6.tar.gz> ^[19].

The original sources (Pythia6409) can be found via Lund FTP server. Note that even in the case when you use your own version of Pythia6 you still need to use the following 2 small files to build the shared library:

```
pythia6_common_address.c
tpythia6_called_from_cc.F
```

For more details on the build procedure, see an example in the file **makePythia6.linux**.

More information about Pythia8 is available from: <http://www.thep.lu.se/~torbjorn/Pythia.html> ^[20].

GSL

The GNU Scientific Library, GSL is used in the mathmore package of ROOT. To install GSL package see <http://www.gnu.org/software/gsl/> ^[21] and configure ROOT with, e.g.:

```
--with-gsl-incdir="the directory where gsl/gsl_version.h is"
```

```
--with-gsl-libdir="the directory where the gsl library is"
```

Also when building GSL on 64 bit machines, like Mac OS X 10.5, configure GSL like:

```
./configure CFLAGS=-m64 --with-pic
```

FFTW

Only a basic interface to FFTW v3 is implemented (no "advanced" or "guru" features). To install this package see <http://www.fftw.org> [22] and configure ROOT with, e.g.:

```
--with-fftw3-incdir="the directory where fftw3.h is"
--with-fftw3-libdir="the directory where the fftw library is"
```

Also when building FFTW on 64 bit machines, like Mac OS X 10.5, configure FFTW like:

```
./configure CFLAGS=-m64 --with-pic
```

graphviz

The *graphviz* package is used by the `TGraphStruct` class to represent structural information as diagrams of abstract graphs and networks.

This package can be downloaded from <http://www.graphviz.org/> [23].

To find *graphviz* the ROOT's configure file looks in standard locations. It is possible to define a specific location using the configure flags:

```
--with-gviz-incdir="the directory where gvc.h is"
--with-gviz-libdir="the directory where the libgvc library is"
```

To install *graphviz* from the sources it is recommended to use the following configure flags:

```
--enable-static=yes --enable-shared=no --with-pic --prefix="graphviz installed here"
```

On 64 bits machines, the ROOT sources are compiled with the option `-m64`. In that case *graphviz* should be also compiled in 64 bits mode. It might be the default option, but on some machine it is not. In that case the environment variable `CC` should be defined as:

```
CC="gcc -m64"
```

before doing `configure`.

On Windows machines it is recommended to not install *graphviz* but to download the pre-installed version from <http://www.graphviz.org/> [23]. The ROOT configure command remains the same.

Bonjour/Zeroconf

To get support of Bonjour/Zeroconf you must install the appropriate libraries for your platform and compile enabling the Bonjour support. See the [PROOF configuration pages](#) [24] for details.

© 1995-2013 The ROOT Team

Source URL: <http://root.cern.ch/drupal/content/installing-root-source>

Links:

- [1] <http://root.cern.ch/drupal/category/package-context/building>
- [2] <http://root.cern.ch/drupal/category/package-context/compiling>
- [3] <http://root.cern.ch/drupal/category/package-context/installing>
- [4] <http://root.cern.ch/drupal/category/package-context/source>
- [5] <http://root.cern.ch/drupal/downloading-root>
- [6] <http://root.cern.ch/drupal/nightlies>
- [7] <http://root.cern.ch/drupal/content/subversion-howto>
- [8] <http://root.cern.ch/svn/root/trunk>
- [9] <http://root.cern.ch/svn/root/tags/v5-26-00b>
- [10] <http://root.cern.ch/drupal/content/build-prerequisites>
- [11] <http://www.cmake.org/>
- [12] <http://root.cern.ch/drupal/building-root-cmake>
- [13] <http://root.cern.ch/lxr/data/root/README/BUILDSYSTEM>
- [14] <http://savannah.cern.ch/bugs/?func=additem&group=savroot>
- [15] <http://www.mesa3d.org>
- [16] <http://root.cern.ch/drupal/how-access-root-files-remotely-rfio>
- [17] http://savannah.cern.ch/files/index.php?group=castor&thread_max=2&highlight=

- [18] <http://www.mysql.com>
- [19] <ftp://root.cern.ch/root/pythia6.tar.gz>
- [20] <http://www.thep.lu.se/%7Etorbjorn/Pythia.html>
- [21] <http://www.gnu.org/software/gsl/>
- [22] <http://www.fftw.org>
- [23] <http://www.graphviz.org/>
- [24] <http://root.cern.ch/drupal/content/configuration-faq#bonjour>