- blender.org
- code.blender.org

Dev:Doc/Building Blender/Linux/Ubuntu/CMake

Log in
< Dev:Doc | Building Blender | Linux | Ubuntu

Blender 2.6

- Blender 2.6
- Blender 2.5
- **Blender 2.4**

English

- Arabic
- Bulgarian
- Catalan
- Czech
- German
- Danish
- **English**
- Greek
- Esperanto
- Spanish
- Estonian
- Farsi

- Finnish
- French
- Indonesian
- Italian
- Japanese
- Korean
- Lithuanian
- Macedonian
- Mongolian
- Dutch
- Polish
- Portuguese

- Romanian
- Russian
- Serbian
- Swedish
- Thai
- Turkish
- Ukrainian
- Chinese

- Developer Documentation page
- Discussion
- View source
- History

Page

- What links here

- Related changes
- Permanent link

From BlenderWiki

Here you find information about how to build Blender on modern machines and distributions.

Troubleshooting
In case you're trying to build on old distributions, chances are that your package manager doesn't provide a recent Python, Collada, etc: in those special cases please have a look at the troubleshooting page

To have latest Blender successfully running on Linux system you follow few simple steps.

This is not as hard as most people would think if you follow these instructions.

1. Install required package dependencies.
2. Download Blender source.
3. Compile Blender.

Get the source

The first step is to get the latest Blender source code from blender.org's GIT repository.

Copy and paste the following instructions into a terminal window. The following commands create a `blender-git` folder in your home directory by downloading the latest source code commonly referred to as `'master'`. An Internet connection is needed.

```
mkdir ~/blender-git
cd ~/blender-git
git clone http://git.blender.org/blender.git
cd blender
git submodule update --init --recursive
git submodule foreach git checkout master
git submodule foreach git pull --rebase origin master
```

If you want to update your git clone checkout to the latest source do (in ~/blender-git/blender/):

```
git pull --rebase
git submodule foreach git pull --rebase origin master
```

For additional information on using Git with Blender's sources, see: Tools/Git

Install the dependencies

Automatic dependencies installation

The preferred way to install dependencies under Linux is now to use the `install_deps.sh` script featured with Blender sources. It currently supports Debian (and derived), Fedora, Suse and Arch distributions. When using the `install_deps.sh` script, you are only required to install the following dependencies:

```
sudo apt-get update; sudo apt-get install git build-essential
```

Then, get the sources and run `install_deps.sh` as explained below.

To build with FFMPEG and Cycles enabled, various libraries are needed. These can be…

- Installed through your distributions repositories.
- Built manually.
- Skipped (many dependencies are optional).

- Managed by a script which checks for missing dependencies and installs them or compiles from sources.

The later solution is now the preferred one! Simply run:

```
cd ~/blender-git
./blender/build_files/build_environment/install_deps.sh
```

This script works for Debian/Redhat/SuSE based distributions, both 32 and 64 bits.

This scripts accepts some optional command lines arguments (use `--help` one to get an exhaustive list), among which:

`--source <path>`
: Where to store downloaded sources for libraries we have to build (defaults to `~/src/blender-deps`).

`--install <path>`
: Where to install the libraries we have to build (defaults to `/opt/lib`).

`--with-osl`
: Try to build OpenShadingLanguage. Note that this is still considered as experimental!

`--skip-osl`
: Skip download and build OpenShadingLanguage.

Some commands in this script requires sudo, so you'll be likely asked couple of times for your password.

When the script finish installing/building all the packages, it'll print which parameters for CMake and SCons you should use to use compiled libraries.

If you have to compile your own boost libraries, you will have to tell Linux where to find them to get blender running. Just run the following commands as root user, from a terminal window:

```
echo "/opt/lib/boost/lib" > /etc/ld.so.conf.d/boost.conf
ldconfig
```

Manual dependencies installation

To manually install Blender's dependancy packages:

```
sudo apt-get update; sudo apt-get install git build-essential \
 libx11-dev libxi-dev \
 libsndfile1-dev \
 libpng12-dev libjpeg-dev libfftw3-dev \
 libopenexr-dev libopenjpeg-dev \
 libopenal-dev libalut-dev libvorbis-dev \
 libglu1-mesa-dev libsdl1.2-dev libfreetype6-dev \
 libtiff5-dev libavdevice-dev \
 libavformat-dev libavutil-dev libavcodec-dev libjack-dev \
 libswscale-dev libx264-dev libmp3lame-dev python3.4-dev \
 libspnav-dev libtheora-dev libglew-dev
```

As a final note, here are the key libraries that you may want to use with Blender:

Python (http://www.python.org/) (3.4)
: Needed for interface scripts (building without Python is supported but not meant for general use).

Boost (http://www.boost.org/) (min 1.49)
: Necessary for Cycles, OSL, Audaspace, Internationalization…

OpenColorIO (http://opencolorio.org/) (min 1.0)
: Necessary to handle color spaces.

OpenImageIO (https://sites.google.com/site/openimageio/)
: Necessary for Cycles and OSL (min 1.1 in this case).

LLVM (http://llvm.org/) (min 3.0)
: Necessary for OSL.

OpenShadingLanguage (http://code.google.com/p/openshadinglanguage/)
: Enable custom shaders under Cycles.

FFMPEG (http://ffmpeg.org/) or libav (ffmpeg fork) (http://libav.org/)
: For handling most video formats/codecs.

Compile Blender with CMake

Installing CMake

From within your package manager, install:

- **cmake**
- a cmake's configuration tool like
  - **ccmake** (text based interface) or,
  - **cmake-gui** (GUI configuration tool).

Automatic CMake Setup

If you're not interested in manually setting up CMake build directory, configuring, building and installing in separate steps, we provide a convenience makefile in blenders source directory which sets up cmake for you.

```
cd ~/blender-git/blender
make
```

Updating blender is as simple as:

```
cd ~/blender-git/blender
git pull --rebase
git submodule foreach git pull --rebase origin master
make
```

Once the build finishes you'll get a message like..

```
blender installed, run from: /home/me/blender-git/build_linux/bin/blender
```

There are some pre-defined build targets:

- **`make`** - some are turned off by default because they can be difficult to correctly configure for newer developers and aren't essential to use & develop Blender in most cases.
- **`make lite`** - the quickest way to get a Blender build up & running, can also help to avoid installing a lot of dependencies if you don't need video-codecs, physics-sim & cycles rendering.
- **`make full`** - this makes a complete build with all options enabled, matching the releases on **blender.org**.

For a full list of the optional targets type...

```
make help
```

Manual CMake Setup

If you want to have more control over your build and have configuration, building and installation as separate steps or have multiple build directories for a single source dir. You can follow these steps.

Preparing CMake's directory

Let's suppose that you've checked out Blender's source (http://git.blender.org/gitweb/gitweb.cgi/blender.git) in the folder `~/blender-git/blender`.

Please note that in some cases the "blender" directory may have a different name. Some branches are modified copies of the *trunk/blender* directory and are named e.g. soc-2009-name_of_participant.

Now you have to choose a location for CMake build files. In CMake you could do an *"in-source" build* or an *"out-of-source"* build, but

currently **in-source builds in Blender are not allowed**.

Out-of-source build

By doing an "out-of-source" build (http://www.cmake.org/Wiki/CMake_FAQ#What_is_an_.22out-of-source.22_build.3F) you create a CMake's folder aside from `~/blender-git/blender`, for example `~/blender-git/build`:

```
mkdir ~/blender-git/build
cd ~/blender-git/build
cmake ../blender
```

This will generate makefiles in the build folder (`~/blender-git/build`).

Warning

As said above, *in-source-builds* where you build blender from the source code directory are not supported.

If CMake finds a `CMakeCache.txt` in the source code directory, it uses that instead of using `~/blender-git/build/CMakeCache.txt`.

If you have tried to do an in-source build, you should remove any CMakeCache.txt from the source code directory before actually running the out-of-source build:

```
rm -f ~/blender-git/blender/CMakeCache.txt
```

Editing CMake Parameters

Note that CMake should detect correct parameters so you shouldn't need change defaults to simply to compile blender, this is only if you want to change defaults, make a debug build, disable features etc,
so you may want to skip this section for now and come back to it later if you want to make adjustments.

You can modify the build parameters in different ways:

- editing `~/blender-git/build/CMakeCache.txt` file in a text editor
- using **cmake-gui** if your distro supports it

  ```
  cmake-gui ../blender
  ```

  Opens a graphical interface where you can easily change parameters and re-configure things.

- using **ccmake** tool, with

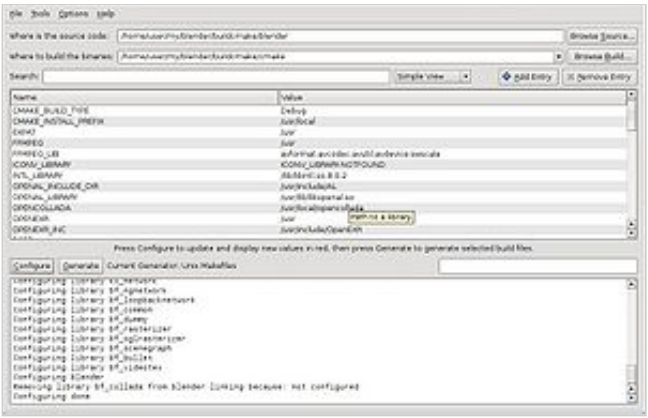
  ```
  ccmake ../blender
  ```

  Opens a text interface where you can easily change parameters and re-configure things.

- cmake parameters can also be easily set on the command line, for eg.

```
cmake ../blender \
    -DCMAKE_INSTALL_PREFIX=/opt/blender \
    -DWITH_INSTALL_PORTABLE=OFF \
    -DWITH_BUILDINFO=OFF \
    -DWITH_GAMEENGINE=OFF
```

These commands are exactly those found in `~/blender-git/build/CMakeCache.txt` so you can copy commands from there and use them in the command line without running ccmake.



Screenshot from cmake-gui, qt based gui for cmake

Screenshot from ccmake, curse-based gui for cmake

Building Blender

After changes have been done and you have generated the makefiles, you can compile using the *make* command inside the build folder:

```
cd ~/blender-git/build
make
make install
```

Parallel Builds

For multi-core / multi processor systems you can build much faster by passing the jobs argument to make: -j(1+*number_of_cores*).

For example put "**-j3**" if you have a dual-core or "**-j5**" if you have a quad-core.
For Shell-scripts use GNU-Syntax: expr `nproc` + 1 or Bash-Syntax: $((`nproc`+1))

If it's still not using all your CPU power, leave out the number: "**-j**" to spawn as many as possible build processes (usually 64).
Note that it may build faster, but make your system lag in the meanwhile.

Also notice the **install** target is used, this will copy scripts and documentation into **~/blender-git/build/bin**

For future builds you can simply update the repository and re-run make.

```
cd ~/blender-git/blender
git pull --rebase
git submodule foreach git pull --rebase origin master
cd ~/blender-git/build
make
make install
```

Notice

Both portable & system installations are supported.

Portable installation is default where scripts and data files will be copied into the build '~/blender-git/build/bin' directory and can be moved to other systems easily.

Disable `WITH_INSTALL_PORTABLE` to install into `CMAKE_INSTALL_PREFIX` which uses a typical Unix FHS (http://en.wikipedia.org/wiki/Filesystem_Hierarchy_Standard) layout: bin/, share/blender/, man/ etc.

Static Linking

If you want to share your build with others (for example through graphicall (http://graphicall.org)), you should statically link against as much libraries as possible.

Listing needed shared libs
You can see all needed shared libraries by simply running `ldd path/to/blender`! Nice and easy!

Note: by default, Python is not static. To make it so, You'll have to edit `PYTHON_LYBRARY` to something like `/usr/lib/x86_64-linux-gnu/libpython3.3m.a;/usr/lib/x86_64-linux-gnu/libexpat.a` (exact paths depend on your OS).

Static Linking Each Library:

Enable general static linking
: Set WITH_STATIC_LIBS to ON, so that CMake will try to make blender link against static libs, when available. Though not perfect, this should reduce quite nicely needed shared libs.

ffmpeg
: Set the FFMPEG_LIBRARIES var to full paths of all needed libs (xvid, x264, etc.).

boost
: Set (or create) Boost_USE_STATIC_LIBS to true. If you are using boost automatically built by the install_deps.sh script, you should not need to do anything else.
But if you are using your own distro's boost package, you most likely will have to set (or create) Boost_USE_ICU to true too (and install the relevant libicu development package), to get blender linking.

TODO: other libs?

See also

CMake

- Introduction to CMake (http://playcontrol.net/ewing/screencasts/getting_started_with_cmake_.html)
- Bill Hoffman presents CMake at GoogleEdu (http://www.youtube.com/watch?v=8Ut9o4OdSC0#t=07m41s) (starts at 7:41, relevant to cmake until 31:18)

IDE setups with CMake

- How to compile Blender using CMake with QTCreator on Linux

Categories: Syntax highlight | Script

Contents

- Popular pages
- New files
- New pages
- Recent changes