Building LibreOffice on Linux: Tips and Tricks

De The Document Foundation Wiki
< Development(Redirigido desde «Development/How to build»)

TDF (http://www.documentfoundation.org/) LibreOffice (http://www.libreoffice.org/) Community Blogs (http://planet.documentfoundation.org/) Pootle (https://translations.documentfoundation.org/) Moztrap (http://manual-test.libreoffice.org/) ODFAuthors (http://www.odfauthors.org/libreoffice/english/) Owncloud (https://owncloud.documentfoundation.org/common) Redmine (https://redmine.documentfoundation.org) Box (http://www.libreofficebox.org/) Ask LibreOffice (http://ask.libreoffice.org) | Donate (http://donate.libreoffice.org//)

Home | Development | Design | QA | Events | Documentation | Website | Localization | Marketing | LibreOffice-Box | Macros | | | Wiki Help

Overview | Reporting Bugs | How to build | Code Overview | Git Commands | Debugging | Easy Hacks | Release Plan | FAQ | Developers | || | →Open Issues (https://bugs.libreoffice.org/report.cgi?x_axis_field=bug_severity&y_axis_field=component&z_axis_field=&query_format=report-table&short_desc_type=allwordssubstr&short_desc=&product=LibreOffice&bug_status=UNCONFIRMED&bug_status=NEW&bug_status=ASSIGNED&bug_status=REOPENED&bug_status=RES
--
&longdesc_type=allwordssubstr&longdesc=&bug_file_loc_type=allwordssubstr&bug_file_loc=&status_whiteboard_type=allwordssubstr&status_whiteboard=&keywords_type=allwords&keywor

EN | AN | AR | AST | BE | BG | BN | BRX | CA | CA-VAL | CS | DA | DE | EL | EO | ES | FA | FI | FR | GD | GL | HE | HI | HU | ID | IS | IT | JA | JV | KO | LO-LA | LT | MR | NL | NO | OC | OM | PA | PT | PT-BR | RO | RU | SAH | SK | SL | SV | TE | ไทย (TH) | TR | UK | VI | 中文正體 (ZH-TW) | 中文简体 (ZH-HANS)

This article is about building on Linux. For other platforms see Building on Mac OS X or Building on Windows

Contenido

Building LibreOffice on Linux

Build dependencies

In general, the easiest way to build LibreOffice is on Linux.

First you need to install the build dependencies:

```
On Debian/Ubuntu: sudo apt-get build-dep libreoffice
On OpenSUSE 11.4+: sudo zypper si -d libreoffice
On Fedora 15+ & derivatives: sudo yum-builddep libreoffice
```

Note that these commands might not install everything that is necessary, because they use the distro's package as a source of dependencies. That means they reflect the version of LibreOffice that is packaged by the distro and the options that are used for building it. For more details or information for other distros such as Arch, see the page on build dependencies on Linux.

Cloning and building

Next, we `clone` the repository and start to build:

```
$ git clone git://anongit.freedesktop.org/libreoffice/core libreoffice
$ cd libreoffice
$ ./autogen.sh
$ make
```

See notes below for additional options to pass to autogen (e.g. extra debug support).

Running your build

This will create you a good local installation, which you can then start with:

```
$ instdir/program/soffice --writer
```

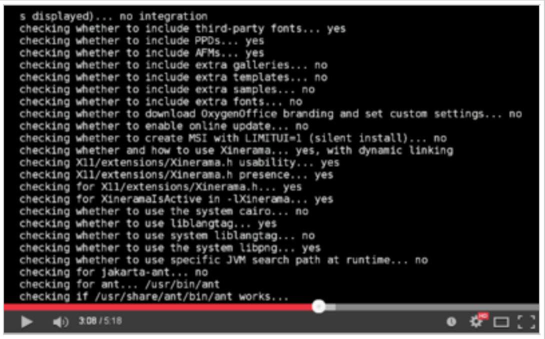
directly under a debugger. Then you can open writer with the command:

```
(gdb) run -writer
```

Note that *soffice.bin* will terminated the first time it is run directly (or via make debugrun), this is normal, you simply need to run it again to make use of it. Running *soffice* will deal with this for you.

Video Tutorials

First Build Video Tutorial



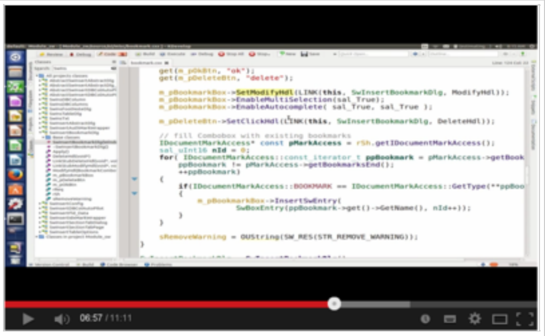
```
s displayed)... no integration
checking whether to include third-party fonts... yes
checking whether to include PPDs... yes
checking whether to include AFMs... yes
checking whether to include extra galleries... no
checking whether to include extra templates... no
checking whether to include extra samples... no
checking whether to include extra fonts... no
checking whether to download OxygenOffice branding and set custom settings... no
checking whether to enable online update... no
checking whether to create MSI with LIMITUI=1 (silent install)... no
checking whether and how to use Xinerama... yes, with dynamic linking
checking X11/extensions/Xinerama.h usability... yes
checking X11/extensions/Xinerama.h presence... yes
checking for X11/extensions/Xinerama.h... yes
checking for XineramaIsActive in -lXinerama... yes
checking whether to use the system cairo... no
checking whether to use liblangtag... yes
checking whether to use system liblangtag... no
checking whether to use the system libpng... yes
checking whether to use specific JVM search path at runtime... no
checking for jakarta-ant... no
checking for ant... /usr/bin/ant
checking if /usr/share/ant/bin/ant works...
```

Getting your first LibreOffice build (https://www.youtube.com/watch?v=2gIqOOajdYQ&hd=1)

While this introduction was done on an Ubuntu system, it should work with little modification on any Linux system (see below for details, if needed). We would recommend to watch the video along with performing the steps -- it not only walks you through the setup, it also has a excellent soundtrack! ;)

Working on LibreOffice from an IDE

The following video shows how you can work on LibreOffice from an IDE when you do `make kdevelop-ide-integration`:



```
    get(m_pOkBtn, "ok");
    get(m_pDeleteBtn, "delete");

    m_pBookmarkBox->SetModifyHdl(LINK(this, SwInsertBookmarkDlg, ModifyHdl));
    m_pBookmarkBox->EnableMultiSelection(sal_True);
    m_pBookmarkBox->EnableAutocomplete( sal_True, sal_True );

    m_pDeleteBtn->SetClickHdl(LINK(this, SwInsertBookmarkDlg, DeleteHdl));

    // fill Combobox with existing bookmarks
    IDocumentMarkAccess* const pMarkAccess = rSh.getIDocumentMarkAccess();
    sal_uInt16 nId = 0;
    for( IDocumentMarkAccess::const_iterator_t ppBookmark = pMarkAccess->getBook
        ppBookmark != pMarkAccess->getBookmarksEnd();
        ++ppBookmark)
    {
        if(IDocumentMarkAccess::BOOKMARK == IDocumentMarkAccess::GetType(**ppBoo
        {
            m_pBookmarkBox->InsertSwEntry(
                    SwBoxEntry(ppBookmark->get()->GetName(), nId++));
        }
    }

    sRemoveWarning = OUString(SW_RES(STR_REMOVE_WARNING));
}
```

Working on LibreOffice from an IDE (https://www.youtube.com/watch?v=-5hVXeHNt2M&hd=1)

A shorter teaser version of this video (https://www.youtube.com/watch?v=Shdfi_RKb8s) (for sharing on social media etc.) is also available.

Details, Tips, Tricks and Hints

Only read beyond this point if you look for various ways of optimizing or troubleshooting your build. Note the tips below are a wild collection of observations that might or might not apply to you.

Preparation

Dependencies

There exists a shell script that download all dependencies and the source code in some Linux distros. It can be find in the github, the name is pre-install.sh: https://github.com/marcosps/lo_useful.git

This page documents how to build the current development version of LibreOffice, based on a git repository named 'core'.

See also Development/Linux Build Dependencies

Disk Space

At minimum, with the git repositories, and the product built and packaged, you will need 26GB of disk space, depending on the platform, the build options and the specific autogen options you use, a few Gb more if you make a debug build.

If you are working on multiple features at once you can either use git stash or you can setup more than one working directory (we will see later how to set them up). A workdir allow you to have a separate build environment without having to duplicate the git history, but it will still cost you 6 to 9 Gb per workdir.

Getting the sources

In the rest of this tutorial we will work in ~/git. Of course you are free to choose which ever directory you want as a starting point. Now let's download the 'core' git repository:

```
$ git clone git://anongit.freedesktop.org/libreoffice/core libreoffice
Cloning into libreoffice...
Remote: Counting objects: 76845, done.
remote: Compressing objects: 100% (17328/17328), done.
remote: Total 76845 (delta 60786), reused 74045 (delta 58579)
Receiving objects: 100% (76845/76845), 15.82 MiB | 1.17 MiB/s, done.
Resolving deltas: 100% (60786/60786), done.
$ cd libreoffice
```

Autogen

Now you need to configure your build. This is done by running a script named 'autogen.sh'. There are many options that can be given to that scripts, please refer to this **page describing the options** for information.

After you run autogen.sh the first time, you can also get a list of the options by running:

```
$ ./autogen.sh
[long output]
$ ./autogen.sh --help
'configure' configures LibreOffice 4.0 to adapt to many kinds of systems.

Usage: ./configure [OPTION]... [VAR=VALUE]...

To assign environment variables (e.g., CC, CFLAGS...), specify them as
VAR=VALUE.  See below for descriptions of some of the useful variables.

Defaults for the options are specified in brackets.

Configuration:
  -h, --help              display this help and exit
      --help=short        display options specific to this package
      --help=recursive    display the short help of all the included packages
[....]
```

You mileage may vary, depending on your platform and distribution. Here are a few examples:

```
$ ./autogen.sh --enable-dbgutil --without-java --without-help --without-myspell-dicts # used on linux
$ ./autogen.sh --enable-dbgutil # used on MacOS
$ ./autogen.sh --enable-dbgutil --without-help --without-myspell-dicts # without external repositories (only core)
```

The --enable-dbgutil option configures for a developer build, disabling optimizations, enabling debugging support and additional checks.

Building on Linux Tips

see platform-independant tips at Development/GenericBuildingHints

Start what you have built

To run the LibreOffice you have just built, do

```
$ instdir/program/soffice --writer
```

When you've re-compiled in any way just re-run from instdir/program.

Performance

Building LibreOffice takes time, a lot of time. Exactly how much depends on how powerful your machine is. But there are tools you can use to speed-up things.

ccache

ccache (http://ccache.samba.org/ccache) is a tool that caches the results of c/c++ compilation to try to re-use them later. It will not speed-up your initial build (on the contrary it will slow it down a bit), but it can dramatically speed up later re-building. If you plan to change lots of header files in LibreOffice, it is a worthy tool to have.

By default, ccache will be enabled automatically if it is found on your system. For best results, you may want to increase the cache size or enable cache compression; `man ccache` is your friend.

Note that the default cache size is just 1G which is far too small to be useful for a LibreOffice build; for a build without debug symbols, you should have at least 8G cache, and for a build with debug symbols for everything probably 32G is a useful starting point. You can set the size like this (note that this setting is stored inside the cache directory, if you change the location by setting `$CCACHE_DIR` you have to re-do it):

```
ccache --max-size 32G
```

ccache also supports compression, which is very likely a good idea to enable (does anybody have hard numbers?), especially if you are using a small or slow storage medium, like a SSD or laptop hard disk. You can enable compression by adding this to your `.bashrc` or equivalent:

```
export CCACHE_COMPRESS=1
```

distcc / Icecream

If you are in an environment where you have access to multiple machines with spare cpu cycles, you can take a look at distcc (http://distcc.samba.org/) or icecream (http://people.gnome.org/~michael/blog/icecream.html) , tools for distributed build.

Support for Icecream is built into LibreOffice, it is enough to add `--enable-icecream` to `./autogen.sh`, and the configure process will pre-set number of jobs to use, and will try to find the icecream gcc wrappers in `/opt/icecream/bin`. Should you have them somewhere else, please use `--with-gcc-home=/path/to/your/icecream/bin` switch to override that.

--with-parallelism

The build process can be told to run multiple tasks in parallel. The parallelism is controlled by the autogen parameter --with-parallelism. If you have enough memory, I found that using --with-parallelism=n where n is your number of cores, or 2 for --with-parallelism if you have only 1 core, give me the fastest build time.

--with-parallelism already defaults to the the number of cores/cpus on your system, unless you use --enable-icecream - then to 10.

--with-system-libs

Building with existing libraries may speed up your build, but you have to fiddle with the dependencies.

Approximate times

An Athlon 435 X3 with ccache without java without epm takes about 1.5 hours for the initial make on gentoo.

The first build, on an Intel 2630QM with ccache without java and epm with --with-parallelism at 8 and ext4 formatted disk takes about 1,5 h too on Kubuntu. The following daily build time varies between 2 and 10 min (depending on the number of modified or added files and their complexity).

Multiple Work Dirs

If you want to work on both the master branch, and the current release branch(es) at the same time, you can share git repos, and external source tarballs (note the savings can be significant, especially if you also use the quite huge l10n repo). Note that switching between different release branches within the same working dir is not recommended; even if there are no dependency problems, the amount of stuff you'd have to rebuild is practically equal to a clean rebuild. If you can afford the diskspace, maintaining separate trees is the recommended way.

There are two methods to share git repos across working directories: git clone --reference and git-new-workdir. A referenced clone can avoid re-downloading the repo, but the new workdir will still have its own .git/config, etc. This is supported by upstream git developers. git-new-workdir, however, is not supported and can cause problems if you try to check out the same branch in two working directories, etc.

Setup

Prepare the first build on your disk just like explained above. For the second, and all other builds, do this to clone the initial core repo:

```
$ git clone --reference /path/to/master --branch name-of-release-branch ssh://logerrit/core /dir/to/be/created
```

You'll notice a much quicker clone operation. After that, setup the new work dir with these two extra configure (or autogen.sh) options:

```
$ ./autogen.sh ... --with-referenced-git=/path/to/master --with-external-tar=/path/to/master/src
```

Again, cloning will be much faster. You have to use an absolute path for the --with-external-tar option. After that, you proceed with building just like for the single-workdir case.

The --with-referenced-git option is only needed if you enabled some submodules (translations, dictionaries or help) or you're building the libreoffice-4-0 branch (or one that is even older). On newer branches submodules are disabled by default.

Checking out a release branch

run autogen.sh on master with the specific parameters you need, then run

```
make fetch
```

to fetch the submodules that may be needed depending on the parameters you passed to autogen.sh

then switch branch with

```
 $ ./g checkout -b libreoffice-4-0 origin/libreoffice-4-0
```

rpm not found/ant not found/no package gnome-vfs-2.0

--disable-epm This will fix "rpm not found", as autogen expects to find either dpkg or rpm on the system.

--without-java This will fix "ant not found"

As of the master towards LibreOffice 3.6, to build the UNO SDK (http://api.libreoffice.org) you need to have Doxygen (http://www.doxygen.org) installed so that HTML documentation for the C++ UNO interface can be generated. One option is to install Doxygen from its website (http://www.doxygen.org) and make sure the `doxygen` executable is found on the `PATH`, or specify its exact location via `--with-doxygen=`*`pathname`* configure switch. Another option is to configure `--without-doxygen`, in which case the HTML documentation for the C++ UNO interface will be missing from the UNO SDK being built. A third option is to configure `--disable-odk`, in which case the UNO SDK is not built at all.

Note: Installing 'libgnomevfs2-dev' package on Ubuntu manually will fix "No package 'gnome-vfs-2.0' found" on older release branches. It should not be needed anymore on master by now. Use synaptic package manager or run in terminal

```
 $ sudo apt-get install libgnomevfs2-dev
```

Build DEB and/or RPM Packages

If you are wanting to build DEB or RPM packages, this should do the trick...
RPM and DEB:

```
$ ./autogen.sh --with-distro=LibreOfficeLinux --with-package-format="deb rpm" --enable-epm
$ make
```

RPM:

```
$ ./autogen.sh --with-distro=LibreOfficeLinux --with-package-format=rpm --enable-epm
$ make
```

DEB:

```
$ ./autogen.sh --with-distro=LibreOfficeLinux --with-package-format=deb --enable-epm
$ make
```

Where are the packages?

Look for the files in the ./workdir/installation/ directory. There is a folder for each of the different sets of packages that includes subfolder[s] for the type or types of packages, either DEB, RPM, or both. Inside the subfolder[s] is another folder called install, and it includes two subfolders. One subfolder contains an archive of all of the packages zipped together (the ..._download folder) and the other contains a DEBS/RPMS folder with each individual package. The rest should be self-explanitory!

Dealing with line endings and Git's autocrlf

If autogen.sh produces output like the following:

```
./autogen.sh: line 1: $':\r': command not found
' '--srcdir=/home/user/libreoffice' '--enable-option-checking=fatal'
configure: WARNING: you should use --build, --host, --target
configure: WARNING: invalid host type:
********************************************************************
*
*   Running LibreOffice build configuration.
*
********************************************************************

' not recognizedystem type... Invalid configuration `
 failedre: error: /bin/bash ./config.sub
Error running configure at ./autogen.sh line 241.
```

... then you probably have line ending problems. This might occur if you mistakenly set Git's `core.autocrlf=true`.

To fix the issue, run these commands (**WARNING: uncommitted changes will be lost!**):

```
git config --unset core.autocrlf
git rm --cached -r .
git reset --hard
git clean -x -f
./autogen.sh
```

Obtenido de «http://wiki.documentfoundation.org/index.php?title=Development/BuildingOnLinux&oldid=93619»
Categorías: EN | Linux | Build

Main > Main > Development > Build
Main > Main > Development > Linux
Main > Main > Languages > EN
Main > Main > Languages > EN
Main > Main > Languages > EN