FPLLL

Installation, Compilation, Dependencies

Martin R. Albrecht

2017/07/06

Outline

Build

Dependencies

Build

TL;DR

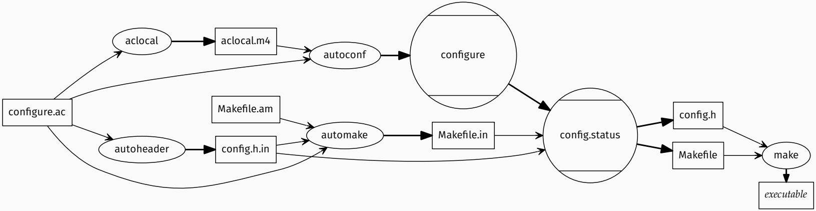
1. `./autogen.sh` (when building from Git)
2. `./configure` (optional: `--prefix=$PREFIX`)
3. `make` (optional: `-jX` for `X` cores)
4. `make check`
5. `make install`

Note

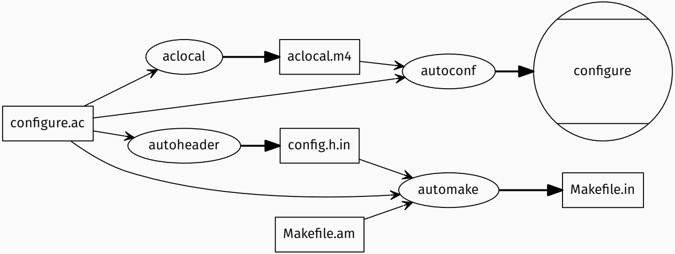
If you used a prefix, you might need to use

```
$ LD_LIBRARY_PATH=$PREFIX/lib fplll …
```

Autotools

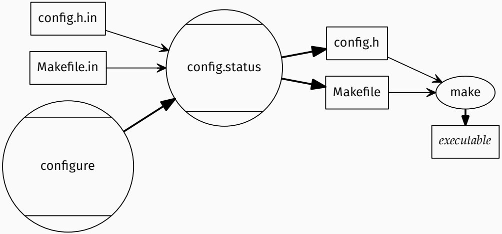

aclocal
aclocal.m4
autoconf
configure
configure.ac
Makefile.am
automake
Makefile.in
config.status
config.h
Makefile
make
autoheader
config.h.in
executable

AUTOTOOLS: `./autogen.sh`



Calling `./autogen.sh`

Autotools: continued



Calling `./configure && make`

See also

https://autotools.io/index.html

Files: `configure.ac`

- configuration options
- finding dependencies and their flags
- version numbers

Versions

There are two version numbers attached to each fplll release:

1. human-readable version number

```
AC_INIT(fplll, 5.1.0)
FPLLL_MAJOR_VERSION=`echo AC_PACKAGE_VERSION | awk -F. '{print $1}'`
FPLLL_MINOR_VERSION=`echo AC_PACKAGE_VERSION | awk -F. '{print $2}'`
FPLLL_MICRO_VERSION=`echo AC_PACKAGE_VERSION | awk -F. '{print $3}'`
FPLLL_VERSION=…
FPLLL_VERSION_NUMBER=…
```

2. Application binary interface (ABI) version number[1]

```
FPLLL_LT_CURRENT=3
FPLLL_LT_REVISION=0
FPLLL_LT_AGE=0
```

[1] This will produce a file `libfplll.so.3.0.0`

ABI Version Number

The version of the libtool library is of the form `current:revision:age`[2]

When doing a release, they should be updated like this:

1. If no interfaces changed, only implementations: just increment revision.
2. If interfaces were added, none removed: increment current, set revision to zero and increment age.
3. If interfaces were removed (breaks backward compatibility): increment current, and set both revision and age to zero.

[2] http://www.gnu.org/software/libtool/manual/html_node/Updating-version-info.html

Files: `Makefile.am`

- which files belongs to which binary
- what files to install in addition to binaries
- one `Makefile.am` per directory

Debugging

1. `CXXFLAGS="-O0 -ggdb -DDEBUG" ./configure`
2. `make V=1` (`V=1` gives more detailed outputs)
3. `make check`
4. `make install` (our tests use the installed `libfplll`

Note

You can also use `./configure --disable-silent-rules` to enable more verbose output when building by default.

Faster Builds

By default, libtool builds everything twice, one for the static and one for the dynamic library.[3] If you want to avoid this double compiling time you can run `./configure --disable-static` which disables building the static library.

[3] https://stackoverflow.com/questions/572760/libtool-slowness-double-building

Prefixes I

It is highly recommended that you do not install `fplll` into your standard path

- It will break, leaving you without a working `fplll`
- Comparing your code with the released code will be useful for debugging
- You may want to compile with debugging flags and without optimisations

Prefixes II

I use Python virtual environments.

1. Creating a new virtual environment

```
virtualenv env
```

2. Using a virtual environment[4]

```
source ./env/bin/activate
export PKG_CONFIG_PATH="$VIRTUAL_ENV/lib/pkgconfig:$PKG_CONFIG_PATH"
export LD_LIBRARY_PATH="$VIRTUAL_ENV/lib"
./configure --prefix="$VIRTUAL_ENV"
```

[4] See `https://github.com/fplll/fpylll` for how to add the exports to `activate`

Adding a new file

- add filename to e.g. `libfplll_la_SOURCES` in `fplll/Makefile.am`
- add header filename to `nobase_include_fplll_HEADERS` in `fplll/Makefile.am`
- add test filename to `tests/Makefile.am`

Dependencies

GMP

`https://gmplib.org`

- used for arbitrary precision integers
- fplll will refuse to compile without it
- used by default, but you can also use machine integers[5]

[5] No idea what difference that makes in terms of performance.

MPFR

http://www.mpfr.org

- used for arbitrary precision floating-point numbers
- fplll will refuse to compile without it
- default is native double precision
- rule of thumb: if you have to use MPFR, you’re dead performance-wise

JSON

`https://github.com/nlohmann/json`

- used to read BKZ strategies
- included in fplll
- could be utilised more for log files etc.

Optional: QD

`http://crd-legacy.lbl.gov/~dhbailey/mpdist/`

- used for higher precision floating-point numbers
- fplll will compile without it
- contains `double double` and `quad double` type
- it seems `quad double` is not faster than MPFR [6]

[6] `https://github.com/fplll/fplll/issues/77`

Fin

Thank You