Building \TeX\ Live (2024)

Peter Breitenlohner
Karl Berry
https://tug.org/texlive
This file documents the \TeX{} Live build system and more.

Copyright © 2016–2024 Karl Berry.
Copyright © 2013–2015 Karl Berry & Peter Breitenlohner.

Permission is granted to make and distribute verbatim copies of this manual provided the copyright notice and this permission notice are preserved on all copies.

Permission is granted to copy and distribute modified versions of this manual under the conditions for verbatim copying, provided that the entire resulting derived work is distributed under the terms of a permission notice identical to this one.

Permission is granted to copy and distribute translations of this manual into another language, under the above conditions for modified versions, except that this permission notice may be stated in a translation approved by the \TeX{} Users Group.
Short Contents

1 Introduction ............................................. 1
2 Overview of build system ............................ 2
3 Prerequisites ........................................... 3
4 Building ................................................. 5
5 Installing ................................................ 11
6 Layout and infrastructure ......................... 14
7 Configure options ..................................... 25
8 Coding conventions .................................. 31
9 Continuous integration .............................. 33
A install-tl ................................................ 35
B tlmgr ...................................................... 46
Index ....................................................... 81
Table of Contents

1 Introduction ................................................. 1

2 Overview of build system ................................. 2

3 Prerequisites ................................................. 3

4 Building ......................................................... 5
   4.1 Build iteration ........................................ 5
   4.2 Build in parallel ..................................... 5
   4.3 Build distribution ..................................... 5
   4.4 Build one package ..................................... 6
   4.5 Build one engine ....................................... 7
   4.6 Cross compilation ..................................... 8
      4.6.1 Cross configuring ................................. 9
      4.6.2 Cross problems ................................... 9

5 Installing ...................................................... 11
   5.1 Installation directories ............................ 11
   5.2 Linked scripts ........................................ 11
   5.3 Distro builds .......................................... 12

6 Layout and infrastructure ................................. 14
   6.1 Build system tools .................................. 14
   6.2 Top-level directories ................................ 14
   6.3 Autoconf macros ...................................... 15
      6.3.1 General setup macros .......................... 15
      6.3.2 Macros for programs ............................ 16
      6.3.3 Macros for compilers ............................ 16
      6.3.4 Macros for libraries ............................ 16
      6.3.5 Macros for library and header flags .......... 17
      6.3.6 Macros for Windows ............................. 17
   6.4 Library modules ...................................... 18
      6.4.1 The png library in libs/libpng ............... 18
      6.4.2 The zlib library in libs/zlib ................ 19
      6.4.3 The freetype library in libs/freetype2 .... 19
      6.4.4 The kpathsea library in texk/kpathsea .... 19
   6.5 Program modules ...................................... 20
      6.5.1 The t1utils package in utils/t1utils ....... 20
      6.5.2 The xindy package in utils/xindy .......... 20
      6.5.3 The xdvik package in texk/xdvik ............ 20
      6.5.4 The subdirectory utils/asymptote .......... 21
Appendix A  install-tl ........................................ 35
A.1 install-tl NAME ........................................ 35
A.2 install-tl SYNOPIS ..................................... 35
A.3 install-tl DESCRIPTION ................................. 35
A.4 REFERENCES ........................................... 35
A.5 install-tl EXAMPLES .................................... 35
A.6 install-tl OPTIONS .................................... 36
A.7 PROFILES ............................................... 41
A.8 ENVIRONMENT VARIABLES ............................. 43
A.9 DIRECTORY TREES .................................... 44
A.10 install-tl BUGS ........................................ 45
A.11 AUTHORS AND COPYRIGHT ............................ 45

Appendix B  tlmgr ............................................. 46
B.1 tlmgr NAME ............................................. 46
B.2 tlmgr SYNOPIS ......................................... 46
B.3 tlmgr DESCRIPTION ................................... 46
B.4 tlmgr EXAMPLES ....................................... 46
B.5 tlmgr OPTIONS ....................................... 47
B.6 ACTIONS ................................................ 50
B.6.1 help .................................................. 50
B.6.2 version ............................................. 50
B.6.3 backup ............................................... 50
B.6.4 bug [search-string] ................................ 51
B.6.5 candidates pkg .................................... 51
B.6.6 check [option...] .................................. 51
   [depends | executes | files | runfiles | texmfdb | all] .................... 51
B.6.7 conf ............................................... 52
B.6.8 dump-tpdb [option...] [–json] ..................... 53
B.6.9 generate .......................................... 53
B.6.10 gui ................................................ 55
B.6.11 info ............................................... 55
B.6.12 init-usertree .................................... 56
B.6.13 install [option...] pkg... ......................... 56
B.6.14 key ................................................ 57
B.6.15 list ............................................... 58
B.6.16 option ............................................. 58
B.6.17 paper ............................................. 59
B.6.18 path ............................................... 60
B.6.19 pinning .......................................... 60
B.6.20 platform ......................................... 61
B.6.21 postaction ........................................ 61
B.6.22 print-platform ................................... 62
B.6.23 print-platform-info ................................ 62
B.6.24 remove [option...] pkg... ......................... 62
B.6.25 repository ........................................ 63
1 Introduction

This manual (dated March 2024) corresponds to the \TeX\ Live 2024 release.

This manual is aimed at system installers and programmers, and focuses on how to configure, build, and develop the \TeX\ Live (TL) sources. It is also available as plain text files in the source tree: `source/README.*`.

The main `source/README` file in the TL source tree provides maximally-terse information for doing a build, and portability information for different systems, along with `source/doc/README.solaris`.

For information on acquiring the TL sources, see `https://tug.org/texlive/svn`. The canonical source repository uses Subversion, and we have no plans to change this.

This manual does not duplicate the information found in other TL documentation resources, such as:

- The \TeX\ Live web pages: `https://tug.org/texlive`.
- The web page describing how to build the binaries which are distributed with \TeX\ Live: `https://tug.org/texlive/build.html`.
- The \TeX\ Live user manual: `https://tug.org/texlive/doc.html`, or run `texdoc texlive`.
- Other \TeX\X-related Texinfo manuals (see Web2c, Kpathsea, etc.): `https://tug.org/texinfohtml/`, or check the ‘\TeX’ category in the GNU Info system.
- Package documentation: `https://tug.org/texlive/Contents/live/doc.html`, or the `doc.html` file at the top level of the installed TL.

As an exception, the full documentation for `install-tl` and `tlmgr` is included here as appendices, simply because it is easy to do so. The same text is available online (linked from `https://tug.org/texlive/doc.html`, or by invoking the program with ‘--help’ (or look at the end of the source).
2 Overview of build system

The \TeX{} Live build system was redesigned in 2009 to consistently use Autoconf, Automake, and Libtool. Thus, running

\texttt{configure \&\& make \&\& make check \&\& make install}

or the essentially-equivalent top-level \texttt{Build} script suffices to build and install the TL programs. The \texttt{make check} clause performs various tests of the generated programs—not strictly required but strongly recommended. Running \texttt{configure --help} will display a comprehensive list of all \texttt{configure} options.

The main components of the TL build system are:

\begin{itemize}
  \item \texttt{libs/lib} \quad Generic libraries.
  \item \texttt{texk/lib} \quad \TeX{}-specific libraries in subdirectories, notably \texttt{lib=kpathsea}. (The other one is \texttt{texk/ptexenc}.)
  \item \texttt{texk/prog} \quad \TeX{}-specific programs (that use Kpathsea).
  \item \texttt{utils/prog} \quad Other programs (that don’t use Kpathsea).
\end{itemize}

The primary design goal of the build system is modularity. Each program and library module (or package) specifies its own requirements and properties, such as required libraries, whether an installed (system) version of a library can be used, \texttt{configure} options to be seen at the top level, and more. An explicit list of all available modules is kept in a single central place: \texttt{m4/kpse-pkgs.m4}.

A second, related goal is to configure and build each library before configuring any other (program or library) module which uses that library. This allows checking for properties and features of a library built as part of the TL tree in much the same way as for a system version of that library.

All generic libraries and several programs are maintained independently. The corresponding modules use (most of) the distributed source tree and document any modifications of that source.

All this is for the sake of simplifying both upgrading of modules and integrating new modules into the TL build system. (Despite all efforts, neither task is easy.)
3 Prerequisites

Overall, building the \TeX\ Live programs, when using all libraries from the TL source tree, requires C and C++11 compilers, GNU \texttt{make}, and Python.

- If \texttt{make} from your \texttt{PATH} is not GNU \texttt{make}, you can set the \texttt{MAKE} environment variable to whatever is necessary.

  GNU \texttt{make} is required only because of third-party libraries, notably FreeType. Automake/Autoconf output in general, and the TL-maintained directories, work with any reasonable \texttt{make}.

- A C++11 compiler is similarly required because of the third-party libraries ICU and HarfBuzz (at least); the program \texttt{dvisvgm} also requires C++11. It is possible to build what remains with older compilers, but you have to remove the C++11-dependent sources. See Section 4.4 [Build one package], page 6.

- Python is required by ICU tests. (If you know how to disable these tests and thus eliminate the requirement, please write.)

A few programs in the tree have additional requirements:

\begin{itemize}
\item \texttt{web2c} requires \texttt{perl} for some tests run by \texttt{make check}. Incidentally, the \TeX\ Live installer (\texttt{install-tl}) and manager (\texttt{tlmgr}) are also written in Perl, but this does not matter for compiling the sources.
\item \texttt{xdvik} \texttt{xpdfopen} require X11 headers and libraries, typically in development packages that are not installed by default.
\item \texttt{xetex} requires \texttt{fontconfig} (again both headers and library), or, for MacOS only, the \texttt{ApplicationServices} and Cocoa frameworks.
\item \texttt{xindy} requires GNU \texttt{clisp}, \texttt{libsigsegv}, and \texttt{libiconv}; additionally, to build the rules and/or documentation: \texttt{perl}, \texttt{latex}, \texttt{pdflatex}.
\end{itemize}

Lacking the required tools, building these programs must avoided, e.g.,

\texttt{configure --without-x --disable-xetex --disable-xindy}

Modifying source files induces more requirements, as one might expect:

- Modification of any \texttt{.y} or \texttt{.l} source files requires \texttt{bison} or \texttt{flex} to update the corresponding C sources.

- Modification of the sources for \texttt{.info} files requires \texttt{makeinfo}.

- Modification of any part of the build system (M4 macros, \texttt{configure.ac}, \texttt{Makefile.am}, or their fragments) requires GNU M4, GNU Autoconf, GNU Automake, and GNU Libtool to update the generated files. Furthermore, to reliably reproduce the build files, the original GNU releases of these tools must be used, not any distro packaging of them. See Section 6.1 [Build system tools], page 14, for more discussion.

\footnote{There is one exception in TL: the \texttt{tangle-sh} and related rules use \texttt{$\$0} to mean the target name, a feature not present in all \texttt{make}s. This could be alleviated by laborious editing, but since there’s no way to avoid GNU \texttt{make} for builds of the entire tree, it does not seem worth the trouble.}
If you haven’t modified any source files, and infrastructure tools such as autoconf or makeinfo are still being run, check your timestamps—notably, use-commit-times must be set to yes in your Subversion configuration (see Section 6.1 [Build system tools], page 14). No infrastructure tools are needed to do a normal build (barring bugs).
4 Building

The top-level Build script is intended to simplify building the binaries distributed with \TeX Live itself—we call this the “native” TL build. It runs configure && make world, which builds everything in a subdirectory of the main source tree (default \texttt{Work/}), installs everything in another subdirectory (default \texttt{inst/}), and finally runs make check. The exact directory and command names can be specified via environment variables and a few leading options. All remaining arguments (assignments or options) are passed to the configure script. Please take a look at the ./Build source file for more information; it is a straightforward shell script.

An alternative, and the one we will mainly discuss here, is to run configure and make in a suitable empty subdirectory. Building in the source directory itself is not supported (sorry).

4.1 Build iteration

Running the top-level configure script configures the top level and the subdirectories \texttt{libs}, \texttt{utils}, and \texttt{texk}. Running make at the top level first iterates over the \TeX-specific libraries, and then runs make in \texttt{libs}, \texttt{utils}, and \texttt{texk} to iterate over the generic libraries, utility programs, and \TeX-specific programs, respectively. These iterations consist of two steps:

1. For each library or program module not yet configured, run configure, adding the configure option \texttt{--disable-build} if the module need not be built, otherwise running make all.
2. For each library or program module that must be built, run make for the selected target(s): \texttt{default} or \texttt{all} to (re-)build, \texttt{check} to run tests, \texttt{install}, etc.

Running the top-level make a second time iterates again over all the library and program modules, and finds (should find) nothing to be done.

4.2 Build in parallel

The TL build system carefully formulates dependencies as well as make rules when a tool (such as tangle, ctangle, and convert) creates several output files. This allows for parallel builds (make \texttt{-j n} with \texttt{n > 1} or even make \texttt{-j}) that can considerably speed up the TL build.

If you’re using TL’s Build script, you can enable \texttt{-j} with the environment variable \texttt{TL\_MAKE\_FLAGS}, as in: ‘env TL\_MAKE\_FLAGS=-j`nproc` ./Build’.

Independently, a noticeable speed-up can also be gained by using a configure cache file, i.e., specifying the configure option \texttt{-C} (recommended).

4.3 Build distribution

Running make dist at the top level creates a tarball \texttt{tex-live-yyyy-mm-dd.tar.xz} from the TL source tree. Running make distcheck also verifies that this tarball suffices to build and install all of TL.

This is useful for checking consistency of the source tree and Makefiles, but the result is not a complete or even usable \TeX system, since all the support files are lacking; see Chapter 5 [Installing], page 11. We do not actually distribute any such tarball, and have no plans to do so.
4.4 Build one package

To build one package, the basic idea is to use the `configure` option `--disable-all-pkgs` (see Section 7.2.2 `--disable-all-pkgs`, page 27). Then all program and library modules are configured but none are made. However, the `Makefiles` still contain all build rules and dependencies and can be invoked to build an individual program or library, first building any required libraries.

Here is an example from start to finish for working on `dvipdfm-x`. Unfortunately, this does not suffice for building the TeX engines; see the next section.

```
mkdir mydir && cd mydir  # new working directory

# Get sources (https://tug.org/texlive/svn), e.g.:
rsync -a --delete --exclude=.svn --exclude=Work \
tug.org::tldevsrc/Build/source/ .

# Create build directory:
mkdir Work && cd Work

# Do the configure:
../configure --disable-all-pkgs --enable-dvipdfm-x \
   -C CFLAGS=-g CXXFLAGS=-g >&outc || echo fail

# Do the make:
make >&outm || echo fail

# Run the tests:
cd texk/dvipdfm-x
make check

# Run the new binary in the build tree, finding support files
# in a separate tree for a TeX Live release YYYY
# (Bourne shell syntax):
TEXMFROOT=/usr/local/texlive/YYYY \
TEXMFCNF=$TEXMFROOT/texmf-dist/web2c \
./xdvipdfmx ...
```

Then you can modify source files in `mydir/texk/dvipdfm-x` and rerun `make` in `mydir/Work/texk/dvipdfm-x` to rebuild; that build directory is where the binary ends up and where you can run a debugger, etc.

The second line of the `configure` invocation shows examples of extra things you likely want to specify if you intend to hack the sources (and not just build binaries): the `-C` speeds `configure` by enabling a cache file, and the `CFLAGS` and `CXXFLAGS` settings eliminate compiler optimization for debugging purposes.

Of course, you need to actually look at the output and check that things are working. There are many `configure` options you can tweak as desired; check the output from `configure --help`. It is also a good idea to run `make check` after making any changes, to ensure that whatever tests have been written still pass.
Reducing source download size

The above retrieves the entire TL source tree (several hundred megabytes). It is natural to ask if this is really necessary. Strictly speaking, the answer is no, but it is vastly more convenient to do so. If you cut down the source tree, you must also give additional `configure` flags to individually disable using system versions of libraries, or the intricacies of the dependencies (such as `teckit` requiring `zlib`) will have undesired side effects. For an example of this approach, see the `build-pdfTeX.sh` script in the `pdfTeX` development source (details at http://pdftex.org), which is indeed such a cut-down TL source tree.

GCC used by default

By default, the `gcc` compilers will be used if present; otherwise, individual packages may use something different. You can explicitly specify the compilers to be used with the environment variables `CC`, `CXX`, and `OBJCXX`.

Removing C++11 dependency

Some libraries and programs require C++11; one such is XeTeX. If you want to build with an older compiler lacking such support, you need to (re)move those source directories; unfortunately, specifying `--disable` for them does not suffice. It’s also necessary to specify `--disable-xetex` explicitly. Specifically, before running `configure --disable-xetex ...`:

```bash
rm -rf libs/icu libs/graphite2 texk/dvisvgm texk/web2c/xetexdir
```

Also, even with `--disable-all-pkgs`, dependencies are (currently) checked. One notable case: if a (non-MacOS) system does not have `fontconfig`, XeTeX cannot be built (see Chapter 3 [Prerequisites], page 3), and `configure` will terminate even with `--disable-xetex`. To proceed without such dependencies, specify `--enable-missing` also.

As of 2022, HarfBuzz also requires C++11. Therefore even more would have to be disabled and removed, notably including `luahtex`, the standard engine used for LuaLaTeX. Removing that would not be acceptable for builds intended for distribution; but perhaps for testing the above information could still be useful.

In general, the TL `configure` will run in all directories. Therefore a general workaround for build problems is to remove failing directories from the tree, and also specify the relevant `--disable-...` option(s).

Patches to improve all this would be most welcome.

4.5 Build one engine

Unfortunately, there is one common case where the steps in the preceding section to build one package (see Section 4.4 [Build one package], page 6) do not suffice: wanting to build one, or a subset, of the TeX engines (or other Web2c programs).

The simplest way to do this is to disable everything and then explicitly specify what to make. For example, to build only the original TeX:

```bash
cd Work       # top build directory
./configure --without-x --disable-shared --disable-all-pkgs \
              --enable-tex --disable-syntex --disable-xetex \
              --enable-missing -C CFLAGS=-g CXXFLAGS=-g
```
make
  cd texk/web2c  # cd engine build directory
make tex        # must specify target

The first `make` run will configure everything, and even build the libraries, even though
the packages are disabled.

The source tree can be cut down to just what is needed for the given engine (the separate
pdfTEX and LuaTEX source repositories do this, for example), but see caveats in previous
section. When the `--disable-xetex` and `--enable-missing` options are needed is also
explained in the previous section.

If you want to debug an X-related program or shared library setup, or other variants,
change the `configure` options accordingly. Either `../Build` or `../configure` can be run.

Then it is necessary to again specify the target engine (`tex`, in the above) in the `make`.

All these complications are rather unfortunate. Patches are welcome.

**Testing one engine**

To run only the tests for a given engine, say hitex:

```
make -C $ww check SUBDIRS=. TESTS='$(hitex_tests)'
```

where `$ww` is the web2c build directory, that is, `ww=/wherever/Build/source/Work/texk/web2c`.

It’s also possible to run individual tests the same way, using the test name exactly as
specified in the `.am` file:

```
make -C $ww check SUBDIRS=. TESTS=hitexdir/tests/hello.test
```

Without the `SUBDIRS=`, errors like this will show up, since `make` will descend into every
directory.

```
fatal: making test-suite.log: failed to create hitexdir/tests/hello.test
```

If you get tired of looking at the ‘Entering’/‘Leaving directory’ lines, you can add the (GNU) make option `--no-print-dir`.

You may find it useful to put lengthy incantations like this into a trivial shell script with
a short name (say, `hitst`). Then you just run `hitst` and edit the file when necessary to
change things around.

### 4.6 Cross compilation

In a cross compilation a build system is used to create binaries to be executed on a host
system with different hardware and/or operating system.

In simple cases, the build system can execute binaries for the host system. This typically
occurs for bi-arch systems where, e.g., i386-linux binaries can run on x86_64-linux
systems and win32 binaries can run on win64 systems. Although sometimes called “native
cross”, technically this is not cross compilation at all. In most such cases it suffices to spec-
yfy suitable compiler flags. It might be useful to add the configure option `--build=host`
to get the correct canonical host name, but note that this should not be `--host=host` (see
Section “Hosts and Cross-Compilation” in GNU Autoconf).

In order to build, e.g., 32-bit binaries with `clang` on a 64-bit MacOS system one could use:

```
TL_BUILD_ENV="CC='clang -arch i386' \`
```
CXX='clang++ -arch i386' \ 
OBJCXX='clang++ -arch i386'' \ 
./Build --build=i386-apple-darwin

4.6.1 Cross configuring

In a standard cross compilation, binaries for the host system cannot execute on the build system and it is necessary to specify the configure options --host=host and --build=build with two different values.

Building binaries requires suitable “cross” tools, e.g., compiler, linker, and archiver, and perhaps a “cross” version of pkg-config and similar to locate host system libraries. Autoconf expects that these cross tools are given by their usual variables or found under their usual name prefixed with host-. Here a list of such tools and corresponding variables:

<table>
<thead>
<tr>
<th>Tool</th>
<th>Variable</th>
</tr>
</thead>
<tbody>
<tr>
<td>ar</td>
<td>AR</td>
</tr>
<tr>
<td>freetype-config</td>
<td>FT2_CONFIG</td>
</tr>
<tr>
<td>g++</td>
<td>CXX</td>
</tr>
<tr>
<td>gcc</td>
<td>CC</td>
</tr>
<tr>
<td>icu-config</td>
<td>ICU_CONFIG</td>
</tr>
<tr>
<td>objdump</td>
<td>OBJDUMP</td>
</tr>
<tr>
<td>pkg-config</td>
<td>PKG_CONFIG</td>
</tr>
<tr>
<td>ranlib</td>
<td>RANLIB</td>
</tr>
<tr>
<td>strip</td>
<td>STRIP</td>
</tr>
</tbody>
</table>

In order to, e.g., build mingw32 binaries on x86_64-linux with a cross compiler found as i386-pc-mingw32-gcc one would specify

```
--host=i386-pc-mingw32 --build=x86_64-linux-gnu
```

or perhaps

```
--host=mingw32 --build=x86_64-linux CC=i386-pc-mingw32-gcc
```

but this latter, especially, might require adding CXX and others.

Configure arguments such as CFLAGS=... refer to the cross compiler. If necessary, you can specify compilers and flags for the few auxiliary C and C++ programs required for the build process as configure arguments

```
BUILDCC=...
BUILDCPPFLAGS=...
BUILDCFLAGS=...
BUILDCXX=...
BUILDCXXFLAGS=...
BUILDLDFLAGS=...
```

4.6.2 Cross problems

The fact that binaries for the host system cannot be executed on the build system causes some problems.

One problem is that configure tests using AC_RUN_IFELSE can compile and link the test program but cannot execute it. Such tests should be avoided if possible and otherwise must supply a pessimistic test result.

Another problem arises if the build process must execute some (auxiliary or installable) programs. Auxiliary programs can be placed into a subdirectory that is configured natively
as is done for texk/web2c/web2c, texk/dvipsk/squeeze, and texk/mdf/xdvik/squeeze. The module books/freetype2 uses the value of CC_BUILD, build-gcc, gcc, or cc as the compiler for the auxiliary program.

The situation for installable programs needed by the build process is somewhat different. A rather expensive possibility, chosen for the ICU libraries in module books/icu, is to first compile natively for the build system and in a second step to use these (uninstalled) programs during the cross compilation.

This approach would also be possible for the tools such as tangle used in the module texk/web2c to build the WEB programs, but that would require first building a native kpathsea library. To avoid this complication, cross compilation of programs written in (C)WEB requires sufficiently recent installed versions of tangle, ctangle, otangle, and tie.

Building xindy requires running the host system clisp binary, thus cross compilation is painful, but possible.
5 Installing

This section discusses the results of make install in the source tree.

The main consideration is that make install is not enough to make a usable TeX installation. Beyond the compiled binaries, (thousands of) support files are needed; just as a first example, plain.tex is not in the source tree.

These support files are maintained completely independently and are not present in the TL source tree. The best basis for dealing with them is the TeX Live (plain text) database in Master/tlpkg/texlive.tlpdb, and/or the TeX Live installer, install-tl. More information is under Master/tlpkg and at https://tug.org/texlive/distro.html.

5.1 Installation directories

Running make install (or make install-strip) installs executables in bindir, libraries in libdir, headers in includedir, general data (including “linked scripts”, see Section 5.2 [Linked scripts], page 11) in datarootdir/texmf-dist, man pages in mandir, and Info files in infodir.

The values of these directories are determined by configure and can be specified explicitly as options such as --prefix=prefix or --bindir=bindir; otherwise, they are given by their usual Autoconf defaults:

- prefix
- exec_prefix
- bindir
- libdir
- includedir
- datarootdir
- mandir
- infodir

except possibly modified as follows:

- If the option --enable-multipath is given, /platform (i.e., the canonical platform name) is appended to bindir and libdir. This is implied for a native TL build.
- In a native TL build, datarootdir is set to prefix, infodir is set to prefix/texmf-dist/doc/info, and mandir to prefix/texmf-dist/doc/man, corresponding to the directories used in the TL distribution.

The top-level configure script displays all these installation paths.

For the native TL build, the Build script leaves the binaries in ./inst/bin/std-system-triplet. The new binaries are not directly usable from that location; they need to be copied to Master/bin/tl-platform. The other files and directories that end up in ./inst/ are ignored.

5.2 Linked scripts

Quite a few executables are architecture-independent shell, Perl, or other interpreted scripts, rather than compiled binaries. A few are maintained as part of the TL source tree, but most are maintained elsewhere with copies under texk/texlive/linked_scripts.
These so-called linked scripts are installed under `datarootdir/texmf-dist/scripts`; for Unix-like systems a symbolic link is made in `bindir`. For example, a symlink points from `bindir/ps2eps` to `datarootdir/texmf-dist/scripts/ps2eps/ps2eps.pl`. For Windows, a standard wrapper binary (copied to, e.g., `bindir/ps2eps.exe`) serves the same purpose. The source for the wrapper is in `texk/texlive/windows_wrapper`.

One reason for this is to avoid having many copies of the same script; a more important reason is that it guarantees the scripts will stay in sync across the different supported operating systems.

Most important of all, we want the `bindir` resulting from the build to be as close as possible to what is in the TL distribution. At present, there are a few exceptions—Asymptote, Biber, Xindy—and each one creates considerable extra work. We don’t want to add more. (See [https://tug.org/texlive/build.html](https://tug.org/texlive/build.html) for information about building those exceptions, as well as the `xz` and `wget` programs that are used in the TL infrastructure.)

### 5.3 Distro builds

Although they use the same code base, building for the native TL distribution as shipped by the TeX user groups is typically quite different from a “distro” build needed by, e.g., a full GNU/Linux or BSD operating system distribution.

The native TL distribution uses shared libraries only when absolutely necessary (`libc`, `libm`, X11 libraries, and `libfontconfig`). In contrast, a distro typically wants to use as many shared libraries as possible from elsewhere on the system, including TeX-specific libraries such as `libkpathsea` (even though Kpathsea has never officially been released as a shared library). In addition, the installation paths will, in general, be completely different.

Here are the `configure` options that distro builds are likely to find most relevant:

```plaintext
--disable-native-texlive-build
   This must be specified to avoid interference from the many tweaks we do for the native TL build.

--with-banner-add=/SomeDistro
   This isn’t technically required, but is strongly recommended, so your build and your distro can be distinguished from others.

--enable-shared
   Build shared versions of the TeX-specific libraries (uses `libtool`).

--disable-static
   Do not build the static versions of the TeX-specific libraries.

--with-system-lib
   Look for and use a system version of the library `lib`. `configure --help` will give you the list of possibilities.

--with-lib-includes=dir
--with-lib-libdir=dir
   If needed, allows you to specify where the headers/code are for the given library `lib`.
```
--prefix=/usr
--prefix=/opt/TeXLive

Or whatever your convention is. The default is /usr/local and you shouldn’t install there for a distro.

--libdir=${exec_prefix}/lib64

May be needed for 64-bit bi-architecture (GNU/Linux) systems.

You will need to take care of the support files mentioned above (see Chapter 5 [Installing], page 11), and many other issues, such as font maps, languages, and formats, independently of the build. Norbert Preining has written a detailed article on adapting TL for distros: https://tug.org/TUGboat/tb34-3/tb108preining-distro.pdf. (If the article needs updating in the future, perhaps we will merge it into this document.)
6 Layout and infrastructure

The TeX Live source tree is the subtree rooted at `Build/source` of the complete TL distribution and contains the sources for all executables distributed by TL, as well as `configure` scripts and `make` rules to build and install them together with some of their support files.

6.1 Build system tools

As mentioned above (see Chapter 3 [Prerequisites], page 3), a normal build has few requirements. On the other hand, if you want to modify the TeX Live infrastructure sources, such as `configure.ac` or `Makefile.am` files, you will need to have several additional tools installed.

In general, the TL build system uses the latest released versions of the GNU build tools, installed directly from the original GNU releases (e.g., by building them with `configure --prefix=/usr/local/gnu` and having `PATH` start with `/usr/local/gnu/bin`). We have found that trying to use the versions of these tools packaged for distros causes many extra hassles, so don’t do that, tempting as it may be.

Currently the versions we use are:

- autoconf (GNU Autoconf) 2.72
- automake (GNU automake) 1.16.5
- bison (GNU Bison) 3.8.2
- flex 2.6.0
- ltmain.sh (GNU libtool) 2.4.7
- m4 (GNU M4) 1.4.19
- makeinfo (GNU texinfo) 7.1

These versions should be used to update the generated files (e.g., `configure` or `Makefile.in`) in all or parts of the TL tree after their dependencies have been changed. This can be done explicitly with the top-level `reautoconf` script or implicitly by using the `configure` option `--enable-maintainer-mode`.

It has often turned out that the bison and flex versions are not critical; however, the autotools versions are. If you don’t have the given versions, get them before modifying the build infrastructure.

The files in the Subversion repository (see https://tug.org/texlive/svn) are all up to date (barring bugs). For this to be reflected by their timestamps in your checkout, be sure to set `use-commit-times=yes` in `~/.subversion/config` or the equivalent.

If timestamps are wrong, you may also be able to avoid unnecessary runs of bison, flex, or makeinfo with `touch` of the generated (.c, .h, or .info) files. With `--enable-maintainer-mode` it may also be necessary to `touch` first `aclocal.m4`, then `configure` and `config.h.in` (or `c-auto.in`), and finally all `Makefile.in` files.

6.2 Top-level directories

Here is a brief description of the top-level directories in the TeX Live source tree.

As mentioned at the beginning of see Chapter 2 [Overview of build system], page 2, the main source directories are `texk/` (TeX-specific programs and libraries), `utils/` (additional programs), and `libs/` (generic libraries).
In addition, the top-level directories `am/` and `m4/` contain `Makefile.am` fragments and Autoconf macros, respectively, used in many places. Specifically, the file `m4/kpse-pkgs.m4` contains lists of all program and library modules; missing modules are silently ignored. (This helps in creating cut-down source trees.)

Each module contributes fragments (in separate files) defining its capabilities and requirements to the `configure.ac` scripts at the top-level and in the subdirectories `libs`, `utils`, and `texk`. The fragments from program modules supply `configure` options to disable or enable building them; those from library modules specify if an installed (system) version of that library can be used. This ultimately determines which modules need to be built—although all modules must be configured for the benefit of `make` targets such as `dist` or `distcheck`.

The top-level `build-aux/` directory contains the common files `compile`, `config.guess`, `config.sub`, `decomp`, etc. used by most packages. These are taken from the GNU Gnulib sources ([https://www.gnu.org/software/gnulib](https://www.gnu.org/software/gnulib)), which in turn synchronizes with any ultimate upstream repository. There are independent copies of some of these in a few other places, e.g., `libs/freetype2/freetype-*/builds/unix/`. The `reautoconf` script does not touch those, but a TL cron job keeps them in sync (nightly).

When the top-level `./Build` script is used to build TL, two more two more top-level directories appear: `Work/` for the build tree, and `inst/` for the install tree (from `make install`). These names (and everything else about `Build`’s operation) can be changed by setting environment variables before running it; see the script source.

### 6.3 Autoconf macros

Here we describe a few of the Autoconf macros used in several modules—many more are defined in the sources; see the top-level `m4/` directory. These general macros are supplemented by module-specific macros in directories such as `texk/dvipng/m4/`; some of those are described in following sections (see Section 6.4 [Library modules], page 18, and Section 6.5 [Program modules], page 20).

#### 6.3.1 General setup macros

The TL sources use two general setup macros:

**KPSE_BASIC**

```
[Macro]

KPSE_BASIC [name, [more-options]]
```

Initialize the basic TL infrastructure for module `name`:
- `AM_INIT_AUTOMAKE([foreign more-options])`
- `AM_MAINTAINER_MODE`
- `KPSE_COMPILER_WARNINGS`
- and make sure the C compiler understands function prototypes. This is used for all generic library and program modules.

**KPSE_COMMON**

```
[Macro]

KPSE_COMMON [name, [more-options]]
```

Like KPSE_BASIC but add:
- `LT_PREREQ([2.2.6])`
- `LT_INIT([win32-dll])`
- `AC_SYS_LARGEFILE`
- `AC_FUNC_FSEEKO`
along with checks for frequently used functions, headers, types, and structures. This is used for \TeX-specific modules.

### 6.3.2 Macros for programs

Macros for program checks:

**KPSE_CHECK_LATEX**

Set LATEX to the first of latex, elatex, or lambda which exists in PATH, or to no if none of them exists. Call AC_SUBST for LATEX. The result of this test can be overridden by setting the LATEX environment variable or the cache variable ac_cv_prog_LATEX.

**KPSE_CHECK_PDFTEX**

Check for pdflatex in PATH and set PDFLATEX.

**KPSE_CHECK_PERL**

Check for perl or perl5 in PATH and set PERL.

**KPSE_PROG_LEX**

Call AC_PROG_LEX and add the flag \texttt{-l} for \texttt{flex}.

### 6.3.3 Macros for compilers

Macros for compiler-related checks:

**KPSE_COMPILER_WARNINGS**

When using the (Objective) C/C++ compiler, set WARNING_[OBJ]C[XX]FLAGS to suitable warning flags (depending on the value given to or implied for \texttt{--enable-compiler-warnings}). Call AC_SUBST for them. At present this assumes GNU compiler warning options, but could be extended to others if necessary.

This macro caches its results in the kpse_cv_warning_cflags, ... variables.

**KPSE_COMPILER_VISIBILITY**

When using the C or C++ compiler, try to set VISIBILITY_C[XX]FLAGS to flags to hide external symbols. Call AC_SUBST for this variable. At present this only tests for the compiler option \texttt{-fvisibility=hidden}, but could be extended if necessary.

This macro caches its results in the kpse_cv_visibility_cflags or kpse_cv_visibility_cxxflags variable.

**KPSE_CXX_HACK**

Provide the configure option \texttt{--enable-cxx-runtime-hack}. If enabled and when using g++, try to statically link with libstdc++, notably improving portability of the resulting binary.

This macro caches its result in the kpse_cv_cxxHack variable.

### 6.3.4 Macros for libraries

One macro for a library check:

**KPSE_LARGEFILE (variable, [extra-define])**

Call AC_SYS_LARGEFILE and AC_FUNC_FSEEKO and append suitable \texttt{-D} flags (optionally including \texttt{-Dextra-define}) to variable.
6.3.5 Macros for library and header flags

Each library module `libs/lib` or `texk/lib` is supplemented by a macro `KPSE_LIB_FLAGS` (all uppercase) that provides make variables for that library. E.g., for `libs/libpng`:

**KPSE_LIBPNG_FLAGS**

[Macro]

Provide the configure option `--with-system-libpng`. Set and `AC_SUBST` make variables for modules using this library (either an installed version or from the TeX Live tree):

- `LIBPNG_INCLUDES` for use in `CPPFLAGS`,
- `LIBPNG_LIBS` for use in `LDADD`,
- `LIBPNG_DEPEND` for use as a Makefile dependency,
- `LIBPNG_RULE` for the `make` rules to rebuild the library.

**KPSE_ADD_FLAGS (name)**

[Macro]

Temporarily extend `CPPFLAGS` and `LIBS` with the values required for the library module `name`.

**KPSE_RESTORE_FLAGS**

[Macro]

Restore `CPPFLAGS` and `LIBS` to their original values.

As an example, the `configure.ac` file for a hypothetical program `utils/foo` using `libpng`, and hence `zlib`, would contain

- `KPSE_ZLIB_FLAGS`  
- `KPSE_LIBPNG_FLAGS`

and its `Makefile.am` would be along these lines:

```plaintext
bin_PROGRAMS = foo  
AM_CPPFLAGS = ${LIBPNG_INCLUDES} ${ZLIB_INCLUDES}  
foo_LDADD = ${LIBPNG_LIBS} ${ZLIB_LIBS}  
foo_DEPENDENCIES = ${ZLIB_DEPEND} ${LIBPNG_DEPEND}  
## Rebuild libz  
@ZLIB_RULE@  
## Rebuild libpng  
@LIBPNG_RULE@
```

If it were necessary to examine whether certain `zlib` or `libpng` features were available, `configure.ac` should be continued this way:

- `KPSE_ADD_FLAGS([zlib])`  
- `KPSE_ADD_FLAGS([libpng])`

6.3.6 Macros for Windows

Windows differs in several aspects from Unix-like systems, many of them due to the lack of symbolic links.
KPSE_CHECK_WIN32 [Macro]
Check if compiling for a Windows system. The result is either no for Unix-like systems (including Cygwin), mingw32 for Windows with GCC, or native for Windows with MSVC. The result is cached in the kpse_cv_have_win32 variable.

KPSE_COND_WIN32 [Macro]
Call KPSE_CHECK_WIN32 and define the Automake conditional WIN32 (true if the value of kpse_cv_have_win32 is not no).

KPSE_COND_MINGW32 [Macro]
Call KPSE_COND_WIN32 and define the Automake conditional MINGW32 (true if the value of kpse_cv_have_win32 is mingw32).

KPSE_COND_WIN32_WRAP [Macro]
Call KPSE_COND_WIN32 and define the Automake conditional WIN32_WRAP (true if the standard Windows wrapper (texk/texlive/windows_wrapper/runscript.exe) exists. This wrapper is used on Windows instead of symlinks for the “linked scripts” (see Section 5.2 [Linked scripts], page 11).

KPSE_WIN32_CALL [Macro]
Call KPSE_COND_WIN32 and check if the file texk/texlive/windows_wrapper/callexe.c exists; if it does, create a symlink in the build tree. Compiling callexe.c with -DEXEPROG="foo.exe" and installing callexe.exe as bar.exe is used on Windows instead of a symlink bar->foo for Unix-like systems.

6.4 Library modules

Here we discuss some specifics for a few of the libraries in TL, both for the details themselves, and as a way of illuminating the general structure and variation.

6.4.1 The png library in libs/libpng

The “generic” png library uses the source tree in the subdirectory libpng-src/, with all modifications for TL recorded in TLpatches/*. The configure.ac fragment ac/withenable.ac contains

KPSE_WITH_LIB([libpng], [zlib])

to specify the module name and indicate the dependency on zlib. A third literal argument ‘tree’ would specify that the library from the TeX Live tree cannot be replaced by a system version. That not being the case here, a second fragment ac/libpng.ac contains

KPSE_TRY_LIB([libpng],
            [#include <png.h>],
            [png_structp png; png_voidp io; png_rw_ptr fn;
             png_set_read_fn(png, io, fn);])

to provide the simple C code

#include <png.h>
int main ()
    { png_structp png; png_voidp io; png_rw_ptr fn;
      png_set_read_fn(png, io, fn);
which Autoconf uses to verify the usability of a system version with C code. The analogous macro `KPSE_TRY_LIBXX` would check using C++. These fragments are included by the `configure.ac` at the top level of TL (`Build/source/configure.ac`).

For this library, like many other modules, a proxy build system for TL is used, consisting of our own `configure.ac`, `Makefile.am`, `include/Makefile.am`; the distributed build system is not used. (Consequently, a few generated files and auxiliary scripts are removed from the distributed source tree.)

The public headers `png.h`, `pngconf.h`, and `pnglibconf.h` are “installed” (as symlinks) under `include/` in the build tree exactly as they are for a system version under, e.g., `/usr/include/`.

The module is supplemented by the file `m4/kpse-libpng-flags.m4` that defines the M4 macro `KPSE_LIBPNG_FLAGS` used by all modules depending on this library in their `configure.ac` to generate the `make` variables `LIBPNG_INCLUDES` for use in `CPPFLAGS`, `LIBPNG_LIBS` for use in `LDADD`, `LIBPNG_DEPEND` for use as dependencies, and `LIBPNG_RULE` for the `make` rules to rebuild the library.

`m4/kpse-libpng-flags.m4` also supplies the configure option `--with-system-libpng`, which then uses `pkg-config` to determine the flags required for the system library.

### 6.4.2 The zlib library in `libs/zlib`

This generic library is very much analogous to `libpng`, but without the dependency on any other library. The file `m4/kpse-zlib-flags.m4` supplies the configure option `--with-system-zlib`, as well as `--with-zlib-includes` and `--with-zlib-libdir` to specify non-standard locations of the `zlib` headers and/or library.

### 6.4.3 The freetype library in `libs/freetype2`

This module uses a wrapper build system. In contrast to the proxy build described earlier, the wrapper build has an almost trivial `configure.ac` and a `Makefile.am` which invokes the `configure` and `make` in the distributed source, followed by `make install` with the TL build tree as destination. In other words, this actually uses the build system provided by upstream (possibly patched).

The flags required for the system library are obtained through `freetype-config`.

### 6.4.4 The kpathsea library in `texk/kpathsea`

This is one of the TeX-specific libraries that are maintained as part of TeX Live (see `Kpathsea` ([tug.org/kpathsea])); the other is `ptexenc`. These TeX libraries are Libtool libraries (static and/or shared) and are installed by `make install` together with the programs. They are, however, not part of the TL DVD as distributed by TeX user groups, and have never been officially released for standalone use.

It is possible, and probably useful for distro builds (see Section 5.3 [Distro builds], page 12), to specify the configure option `--with-system-kpathsea` in order to use a system version of the library. Programs outside the TL tree should use `pkg-config` for the required flags.

In addition to `kpathsea/ac/withenable.ac` and `kpathsea/ac/kpathsea.ac` here there is a third fragment `kpathsea/ac/mktex.ac`, included by both `withenable.ac` and
configure.ac, which supplies configure options such as --enable-mktextfm-default. These determine the compile time default of whether or not to run mktextfm (and similar) to generate a missing .tfm (or whatever) file. In any case, however, the command line options -mktex=tfm or -no-mktex=tfm for the TeX-like engines override this default.

6.5 Program modules

As with libraries (see Section 6.4 [Library modules], page 18), here we discuss the details for a few of the programs in TL.

6.5.1 The t1utils package in utils/t1utils

Here we use the distributed source tree t1utils-src with modifications documented in TLPatches/* and a proxy build system consisting of configure.ac and Makefile.am. The fragment ac/withenable.ac contains

KPSE_ENABLE_PROG([t1utils])

specifying the module name without any dependencies, and supplies the configure option --disable-t1utils.

6.5.2 The xindy package in utils/xindy

This module uses the distributed source tree xindy-src/ with modifications documented in TLPatches/*, and a wrapper configure.ac and Makefile.am that descends into xindy-src.

The xindy build requires a make that supports a VPATH build, can handle all targets, and does not refer to ${top_srcdir} or ${top_builddir}. The fragment xindy/ac/withenable.ac contains

KPSE_ENABLE_PROG([xindy], [kpathsea freetype2], [x])

m4_include(kpse_TL[utils/xindy/ac/xindy.ac])

m4_include(kpse_TL[utils/xindy/ac/clisp.ac])

where disable in the third argument indicates that xindy is only built if explicitly enabled by the user with configure --enable-xindy (the need for clisp makes it too painful to enable by default).

The additional fragments ac/xindy.ac and ac/clisp.ac specify more configure options to be seen at the top level, with ac/xindy.ac also included by configure.ac.

6.5.3 The xdvik package in texk/xdvik

This package is maintained as part of the TeX Live tree with sources in its own directory (texk/xdvik/). The fragment xdvik/ac/withenable.ac contains

dnl extra_dirs = texk/xdvik/squeeze

KPSE_ENABLE_PROG([xdvik], [kpathsea freetype2], [x])

m4_include(kpse_TL[texk/xdvik/ac/xdvik.ac])

thus specifying dependencies on the kpathsea, freetype, and X11 libraries. The M4 comment (following dnl) signals the subsidiary squeeze/configure.ac. This is needed because the main executable xdvi-bin (to be installed as, e.g., xdvi-xaw) is for the host system whereas the auxiliary program squeeze/squeeze has to run on the build system; in a cross compilation, these differ.
The additional fragment `ac/xdvik.ac` is also included by `configure.ac` and supplies the configure option `--with-xdvi-x-toolkit` also seen at the top level.

6.5.4 The subdirectory `utils/asymptote`

This subdirectory contains the sources for `asy` and `xasy` but due to its complexity and prerequisites (e.g., OpenGL) it is not part of the TL build system. These programs must be built and installed independently, but are included on the TL DVD together with their support files. See https://tug.org/texlive/build.html#asymptote.

6.6 Extending TeX Live

This section outlines the basic process for adding new packages to the TL build system.

In any case, a new package directory `foo` should contain the original sources, modified only with changes necessary for TL, in `foo/foo-src`. The changes should be documented in `foo/TLpatches/*`, and also be submitted upstream whenever reasonable. In addition, `foo/` will need the usual Automake build-related files (`configure.ac`, `Makefile.am`, etc.). Please maintain `foo/ChangeLog` for all TL changes.

6.6.1 Adding a new program module

A TeX-specific program module in a subdirectory `texk/prog` may use the TeX-specific libraries and is included by adding its name `prog` to the M4 list `kpse_tekx_pkgs` defined in `m4/kpse-pkgs.m4`.

A generic program module in a subdirectory `utils/prog` must not use the TeX-specific libraries and is included by adding its name `prog` to the M4 list `kpse_utils_pkgs` in `m4/kpse-pkgs.m4`.

In either case, the subdirectory `texk/prog` or `utils/prog` must provide a fragment `ac/withenable.ac` that contains the M4 macro `KPSE_ENABLE_PROG` defined in `m4/kpse-setup.m4` with `prog` as the mandatory first argument and three optional arguments:

1. a list of required libraries from the TL tree;
2. a list of options: `disable` if this module is not to be built without the configure option `--enable-prog`, `native` if cross compilation is not possible, `x` if the program requires X11 libraries;
3. a comment added to the help text for the `configure` option `--enable-prog` or `--disable-prog`.

If the module requires specific `configure` options to be seen at the top level, they should be defined in an additional fragment `ac/prog.ac` included from `ac/withenable.ac` and `configure.ac`.

Usually, the new program is maintained somewhere outside of TeX Live. In that case, as above, we put the upstream sources into a subdirectory `prog-src` (e.g., `utils/newprog/newprog-src`). We do not typically run `configure` in this original `...-src` directory, but only in our own directory; but we do compile using the source files in `...-src`.

So, to summarize the files that must (usually) be created inside a new TL source directory (`texk/newprog` or `utils/newprog`):
ac/withenable.ac
   The KPSE_ENABLE_PROG call just explained.

configure.ac
Makefile.am
   By merging the contents of the original configure.ac (if provided) and a comparable program already in TL. In the above example, one line that will be needed in configure.ac (can be added before the AC_CONFIG_FILES at the end) is:

   AC_SUBST([NEWPROG_TREE], [newprog-src])

   and then use @NEWPROG_TREE@ in Makefile.am where needed.

In general, there is no magic recipe for this part of the job. It’s necessary to think about what needs to be done in the original vs. in \TeX Live. It’s useful to look at the setup for the most comparable programs already in TL that you can find. It’s also useful to grep the entire Build/source tree for whatever you can think of to investigate how something is done. Most of the TL-specific macros are defined in Build/source/m4/*.

TLpatches/TL-Changes
   First actions taken after getting the original source tree; typically removal of derived or unused common files.

TLpatches/patch-...
   If any changes are needed to the original sources, record the patches here so they can be applied next time. Also, send them upstream so that we don’t have to maintain them forever.

ChangeLog
   Record all TL-specific changes, now and in the future.

   After populating the new TL source directory (.../newprog/, in the above), run GNU autoreconf there (see Section 6.1 [Build system tools], page 14). Once that works, if you are the one who’s eventually going to commit the new package, svn add the necessary files, including the generated Makefile.in aclocal.m4 configure, and svn:ignore the Automake cache autom4te.cache. (This is so people checking out the TL source tree do not have to run any autotools, but can simply run configure.)

   To reiterate: do not fail to commit the generated configure and other files. The m4 code in kpse-pkgs.m4 uses the existence of configure to determine whether to descend into (and configure) a given subdirectory.

   Then, run the TL tool reautoconf in the top-level TL Build/source/ directory, to incorporate the new program into the build tree. It is good to then rebuild the whole tree (e.g., using TL’s Build/Build script) to get all the necessary files generated.

   It will probably fail. So then you need to keep at it until the program compiles and tests successfully. The most efficient way is to rerun autoreconf as needed in the new source directory (Build/source/.../newprog), then make in the corresponding build directory (Build/work/.../newprog), then make check, etc. In the end, also make sure that the whole tree builds from scratch.

   After final success, don’t forget to commit. (Or email the TL maintainers with the patch.)
6.6.2 Adding a new engine

Adding a new \TeX{} engine is not completely different from adding a program, but it's not all that similar, either. In this case, the main work is done by creating a new subdirectory of `texk/web2c/` for the engine. The subdirectory is conventionally named ending in `dir`, like `pdftexdir` and `xetexdir`, to avoid clashes with executable names.

The source files for the new engine should be put in this `newenginedir` subdirectory. Also, a file `newenginedir/am/newengine.am` (e.g., `pdftexdir/am/pdftex.am` is needed with the Makefile fragment needed to build it.

The overall `web2c/Makefile.am` needs to have an `include` statement added to insert that `newengine.am` file.

In `web2c/ac/web2c.ac`, a line needs to be added in the definition of the `kpse_tex_progs` variable to include it in the build. That line specifies whether the new engine is built by default, and the additional libraries requires.

For examples of building engines in CWEB, you can check the existing `hitexdir` and `mplibdir` directories; these are somewhat simpler than Lua\TeX{}. Of course, every engine will have its own unique features and requirements, so existing examples will only take you so far.

Web2c is built as one “package”, with each subdirectory’s `.am` fragment inserted with an Automake `include`. This means that, for instance, `$(srcdir)` is `.../web2c`, not `.../webdir/enginedir`. It is a difficult setup to come to terms with, but the alternative is to recurse into each engine subdirectory, and that would be far worse (see Section “Directories” in GNU Automake).

6.6.3 Adding a new generic library module

A generic library module in a subdirectory `libs/lib` must not depend on \TeX{}-specific libraries, by definition. It is included by adding its name `lib` to the M4 macro `kpse_libs_pkgs` in `m4/kpse-pkgs.m4`—before any other libraries from the \TeX{} Live tree on which it depends.

As with program modules, the subdirectory `libs/lib` must contain the sources and build system for the library (and any installable support programs) and a fragment `ac/withenable.ac` that contains the M4 macro `KPSE_WITH_LIB` defined in `m4/kpse-setup.m4` with `lib` as the mandatory first argument and two optional arguments: a list of required libraries from the TL tree, and a list of options: for libraries, currently there is only one—specify `tree` if this library cannot be replaced by a system version.

If a system version can be used, a second fragment `ac/lib.ac` is needed, containing the M4 macro `KPSE_TRY_LIB` (or `KPSE_TRY_LIBXX`) with `lib` as the mandatory first argument and two additional arguments for the Autoconf macro `AC_LANG_PROGRAM` used to compile and link a small C (or C++) program as sanity check for using the system library.

In addition a file `m4/kpse-lib-flags` (at the top level) must define the M4 macro `KPSE_LIB_FLAGS` (all uppercase) setting up the `make` variables `LIB_INCLUDES`, `LIB_LIBS`, `LIB_DEPEND`, and `LIB_RULE` with the values required for `CPPFLAGS`, `LDADD`, dependencies, and a (multi-line) `make` rule to rebuild the library when necessary. All of that is needed for the library from the TL tree and, if supported, for a system version.

If a system library is allowed, `KPSE_LIB_FLAGS` also provides the configure option `--with-system-lib` and uses the additional M4 macro `KPSE_LIB_SYSTEM_FLAGS` to
generate the `make` variables for a system library. In addition, the definition of the M4 macro `KPSE_ALL_SYSTEM_FLAGS` in `m4/kpse-pkgs.m4` must be extended by the line:

```
AC_REQUIRE([KPSE_LIB_SYSTEM_FLAGS])
```

### 6.6.4 Adding a new \TeX-specific library module

A \TeX-specific library module in a subdirectory `texk/lib` may depend on other \TeX-specific libraries but must not depend on any generic library from the TL tree. It is included in the same general ways as a generic library (see the previous section), with these modifications:

- The library name `lib` is added to the M4 macro `kpse_texlibs_pkgs`, which is also in `m4/kpse-pkgs.m4`.
- The fragment `ac/withenable.ac` must use `KPSE_WITH_TEXLIB`. 
7 Configure options

Corresponding to the large number of program and library modules there are a large number configure options, most of which are described here. The command

```
configure --help
```
at the top level gives an exhaustive list of all global options and a few important module-specific ones, whereas, e.g.,

```
texk/lcdf-typetools/configure --help
```
also displays the lcdf-typetools specific options, which are not shown at the top level.

The help text also mentions several influential environment variables, but for \TeX\ Live it is better to specify them as assignments on the command line.

The ./Build script used to make the binaries shipped with \TeX\ Live invokes the top-level configure with a few additional options (see Chapter 4 [Building], page 5). The defaults discussed below are those for the actual configure script; invoking configure via ./Build yields different results.

Defaults for most options are set at the top level and propagated explicitly to all subdirectories. Options specified on the command line are checked for consistency but never modified.

7.1 Global configure options

Here are the global configure options.

7.1.1 --disable-native-texlive-build

If enabled (the default), build for a TL binary distribution as shipped by the \TeX\ user groups. This requires GNU make and implies --enable-multiplatform and --enable-cxx-runtime-hack (unless they are explicitly disabled), and enforces --disable-shared.

If building TL for a GNU/Linux or other distribution, this should be disabled and system versions of most libraries should be used (see Section 5.3 [Distro builds], page 12).

A related option, --enable-texlive-build, is automatically passed to all subdirectories (and cannot be disabled). Subdirectories that can also be built independently from the TL tree (such as utils/xindy and texk/dvipng) but cooperate with TL can use this option to enable TL-specific adaptations, such as installation paths.

7.1.2 --prefix, --bindir, ...

These standard Autoconf options specify various installation directories as usual. For the complete list, see Section 5.1 [Installation directories], page 11.

Also as usual, all values are prefixed by the value of DESTDIR, if set, on the make command line (see Section “Installation in a temporary location” in GNU Automake).

7.1.3 --disable-largefile

Omit large file support (LFS), which is needed on most 32-bit Unix systems for files with 2GB or more. Regardless of this option, the size of DVI and GF files must always be < 2GB, due to the file format specifications.

With LFS, there is no fixed limit on the size of PDF files created by pdftex or PostScript files created by dvips.
7.1.4 --disable-missing
Immediately terminate the build process if a requested program or feature must be disabled, e.g., due to missing libraries. This can help when figuring out a specific (sub)set of modules to enable.

7.1.5 --enable-compiler-warnings=level
Enable various levels of compiler warnings for C, C++, and/or Objective C: the level value can be one of: no min yes max all. The default is yes in maintainer-mode (see below) and min otherwise. This option defines the variables WARNING_[OBJ]C[XX]FLAGS, but these variables are not consistently used in all library and program modules. At present, these warning flags assume options from the GNU compilers.

7.1.6 --enable-cxx-runtime-hack
If enabled (as it is for the native TL build), when using g++, try to statically link with libstdc++, thus improving portability of the resulting binary. See Section 6.3.3 [Macros for compilers], page 16.

7.1.7 --enable-maintainer-mode
Enable make rules and dependencies not useful (and sometimes confusing) to the casual user. This requires current versions of the GNU build tools (see Section 6.1 [Build system tools], page 14), as it automatically rebuilds infrastructure files as needed. See Section “missing and AM_MAINTAINER_MODE” in GNU Automake.

7.1.8 --enable-multiplatform
If enabled (as it is for the native TL build) and --bindir=dir or --libdir=dir are not specified, install executables and libraries in per-platform subdirectories of eprefix/bin and eprefix/lib where eprefix is the value given or implied for exec_prefix. In any case, the values for bindir and libdir are automatically propagated to all subdirectories.

7.1.9 --enable-shared
Build shared versions of the TeX-specific libraries such as libkpathsea. This is not allowed for a native TL build (i.e., --disable-native-texlive-build must also be specified).

7.1.10 --enable-silent-rules
Enable the use of less verbose build rules. When using GNU make (or any make implementation supporting nested variable expansions), you can specify V=1 on the make command line to get more verbosity, or V=0 to get less, regardless of this option.

7.1.11 --without-ln-s
Required when using a system without a working ln -s to build binaries for a Unix-like system. However, make install will not create anything useful, and might fail.

7.1.12 --without-x
Disable all programs using the X Window System.
Chapter 7: Configure options

7.2 Program-specific configure options

Here are (some of) the program-specific configure options.

7.2.1 --enable-prog, --disable-prog

Do or do not build and install the program(s) of module prog.

7.2.2 --disable-all-pkgs

Do not build any program modules by default—only those explicitly enabled. This is useful when one wants to work on only a single program, which is specified with an additional --enable option, e.g., --enable-dvipdfm-x. It's still simplest to check out and configure the whole source tree, but at least only the program you are interested in, and its dependencies, are built. See Section 4.4 [Build one package], page 6.

Without this option, all modules are built except those that are explicitly disabled or specify disable in their ac/withenable.ac fragment.

7.2.3 Configure options for texk/web2c

--with-banner-add=str
Add str to the default version string (which is ‘TeX Live year’ or ‘Web2C year’) appended to banner lines. This is ignored for a native TL build, but distro builds should specify, e.g., /SomeDistro.

--with-editor=cmd
Specify the command cmd to invoke from the e option of TeX and friends, replacing the default vi +%d '%s' for Unix or texworks --position=%d "%s" for Windows.

--with-fontconfig-includes=dir, --with-fontconfig-libdir=dir
Building XeTeX on non-Mac systems requires the fontconfig library headers and code. If one or both of these options are given, the required flags are derived from them; otherwise, they are determined via pkg-config (if present).

--with-mf-x-toolkit
Use the X toolkit (libXt) for Metafont (the default is to use the lowest-level Xlib support; it seems this has the best chance of working across X installations nowadays).

--disable-dump-share
Make the fmt/base dump files architecture dependent (somewhat faster on LittleEndian architectures).

--disable-ipc
Disable TeX's --ipc option.

--disable-mf-nowin
Do not build a separate non-graphically-capable Metafont (mf-nowin).

--disable-tex, --enable-etex, ...
Do not or do build the various TeX, Metafont, and MetaPost engines (defaults are defined in the fragment texk/web2c/ac/web2c.ac).

--disable-web-progs
Do not build the original WEB programs bibtex, ..., weave. Useful if, e.g., you only want to (re)build some engines.
--enable-auto-core
This option causes TeX and Metafont to produce a core dump when a particular hacky filename is encountered, for use in creating preloaded binaries. This is rarely done nowadays.

--enable-libtool-hack
If enabled (which is the default for all platforms), prevents libtool from linking explicitly with dependencies of libfontconfig such as libexpat.

--enable-*win
Include various types of non-X window support for Metafont (EPSF output, mftalk, old graphics terminals, ...).

--enable-tex-synctex, --disable-etex-synctex...
Build the TeX engines with or without SyncTeX support; ignored for a native TeX Live build. Defaults are defined in texk/web2c/ac/web2c.ac.

--disable-synctex
Do not build the SyncTeX library and tool.

7.2.4 Configure options for texk/bibtex-x
The programs bibtex8 and bibtexu have been merged into the module bibtex-x (extended BibTeX).

--disable-bibtex8
Do not build the bibtex8 program.

--disable-bibtexu
Do not build the bibtexu program (building bibtexu requires ICU libraries).

7.2.5 Configure options for texk/dvipdfm-x
The former modules dvipdfmx (extended DVI to PDF converter) and xdvipdfmx (the same, as used by XeTeX) have been merged into dvipdfm-x at the source level. Two separate binaries are still created by default. In addition, dvipdfm is created as a symlink to dvipdfmx, with backward-compatible (very slightly different) behavior.

--disable-dvipdfmx
Do not build the dvipdfmx program or make the dvipdfm symlink.

--disable-xdvipdfmx
Do not build the xdvipdfmx program.

7.2.6 Configure options for texk/dvisvgm
--with-system-libgs
Build dvisvgm using installed Ghostscript (gs) headers and library (not allowed for a native TL build). The default is to load the gs library at runtime if possible, else to disable support for PostScript specials.

--without-libgs
Build dvisvgm without PostScript support at all. Because the dynamic loading just mentioned defeats all attempts at static linking, the result can crash due to library incompatibilities, e.g., on CentOS 5.

--with-libgs-includes=dir, --with-libgs-libdir=dir
Specify non-standard locations of the Ghostscript headers and library.
7.2.7 Configure options for texk/texlive

--disable-linked-scripts
Do not install the “linked scripts” (see Section 5.2 [Linked scripts], page 11), except for the TL scripts required to run texlinks.

7.2.8 Configure options for texk/xdvik

--with-gs=filename
Hardwire the location of Ghostscript (gs) as called by Xdvik.

--with-xdvi-x-toolkit=kit
Use toolkit kit for xdvik, one of: motif xaw xaw3d neXtaw. The default is motif if available, else xaw.

--enable-xi2-scrolling
Use XInput 2.1 “smooth scrolling” if available (default: yes, except for a native TL build).

7.2.9 Configure options for utils/xindy

--enable-xindy-rules
Build and install xindy rules (default: yes, except for a native TL build).

--enable-xindy-docs
Build and install xindy documentation (default: yes, except for a native TL build).

--with-clisp-runtime=filename
Specifies the full path for the Clisp runtime file (lisp.run or lisp.exe) to be installed. When specified as default (the default for a native TL build) the path is determined by the Clisp executable; the value system (not allowed for a native TL build, but the default otherwise) indicates that xindy will use the installed version of clisp (which must be identical to the one used to build xindy).

7.3 Library-specific configure options

Here are (some of) the library-specific configure options, starting with this generic one:

--with-system-lib
Use an installed (system) version of the library lib; this option exists for most libraries, but is not allowed for a native TL build. Using a system version implies also using the system versions of all libraries that lib depends on.

For many libraries --with-lib-includes=dir and --with-lib-libdir=dir can specify non-standard search locations; others use pkg-config or similar to determine the required flags.

The top-level configure script performs a consistency check for all required system libraries and bails out if tests fail.

7.3.1 Configure options for kpathsea

--enable-cmd-default, --disable-cmd-default
Determine the compile time default for whether or not to run cmd, which is one of:

mkocp (Omega compiled translation process file)
mkofm  (Omega font metrics file)
to generate the specified type of file dynamically. The default can be overridden by the user in any case (see Section 6.4.4 [kpathsea library], page 19).

### 7.4 Variables for configure

The values for these variables can be specified as configure arguments of the form \texttt{VAR=value}. They can also be defined in the environment, but that might not work for cross compilations.

- \texttt{CC}
- \texttt{CXX}
- \texttt{CPPFLAGS}

And plenty more. As usual with Autoconf, these variables specify the name (or full path) of compilers, preprocessor flags, and similar. See Section “Preset Output Variables” in GNU Autoconf.

- \texttt{CLISP}

Name (or full path) of the \texttt{clisp} executable, used to build \texttt{xindy}.

- \texttt{FT2_CONFIG}
- \texttt{ICU_CONFIG}
- \texttt{PKG_CONFIG}

These specify the name (or path) for the \texttt{freetype-config}, \texttt{icu-config}, and \texttt{pkg-config} commands used to determine the flags required for system versions of \texttt{libfreetype}, the ICU libraries, and other libraries, respectively.

- \texttt{KPSEWHICH}

Name (or path) of an installed \texttt{kpsewhich} binary, used by make check to determine the location of, e.g., \texttt{cmbx10.tfm}.

- \texttt{MAKE}
- \texttt{SED}

And more. Name (or path) of the \texttt{make}, \texttt{sed}, and similar programs; used at the top level and propagated to all subdirectories.

- \texttt{PERL}
- \texttt{LATEX}
- \texttt{PDFLATEX}

Name (or full path) for the \texttt{perl}, \texttt{latex}, and \texttt{pdflatex} commands used, e.g., to build the \texttt{xindy} documentation.
8 Coding conventions

Ideally, building all of TeX Live with --enable-compiler-warnings=max should produce no (GCC) compiler warnings at all. In spite of considerable efforts into that direction we are still far from that goal and there are reasons that we may never fully reach it. Below are some rules about declarations of functions or variables and the use of const. These rules should be applied to the code maintained in the TeX Live tree and for other packages whose maintainers are willing to accept patches.

8.1 Declarations and definitions

C standards

The TeX Live build system no longer supports pre-ANSI C compilers. Thus all function prototypes and definitions must conform to the ANSI C standard (including void in the declaration of C functions with no parameters). On the other hand, TL is built for a wide variety of systems, not all of which support the C99 standard. Therefore using C99 features should be avoided if that can easily be done. In particular, C code must not contain declarations after statements or C++-style comments.

If some C99 (or later) constructs must be used, the module should verify that they are available and otherwise provide an alternative. For example, the module texk/chktex uses the C99 function stpcpy() that may or may not be available on a particular system. It uses AC_CHECK_DECLS([stpcpy]) in configure.ac to test this, and provides a perhaps less efficient alternative (in the file Utility.h):

```
#if !(defined HAVE_DECL_STPCPY && HAVE_DECL_STPCPY)
static inline char *stpcpy(char *dest, const char *src)
{
    return strcpy(dest, src) + strlen(src);
}
#endif
```

Static functions

Functions used in only one file should be declared static; they require no prototype except in forward declarations.

Extern functions

Functions not declared static, usually because they are used in several files, require an (extern) prototype in exactly one header file, which is included in the file defining the function and in all files using that function—this is the only way to guarantee consistency between definition and use. There should be no extern declarations sprinkled throughout the C code (with or without comments as to where that function is defined).

Variable declarations

The declaration of global variables follows analogous rules: they are either declared static if used in only one file or declared extern in exactly one header and instantiated in exactly one file.
8.2 Const

The `const` feature of C is valuable, but easy to mis-use.

**Function parameters**

Ideally, a function parameter not modified by the function should be declared as `const`. This is important in particular for strings (`char*`) because the actual arguments are often string literals. It is perfectly legitimate and safe to use a type `char*` value for a type `const char*` variable (in an assignment, as initializer, as function argument, or as return value). It is equally safe to use a type `char**` value for a type `const char*const*` variable, but not for a type `const char**` variable since that might cause modification of a quantity supposed to be constant.

Getting all `const` qualifiers right can get quite involved but can almost always be done. There are only a couple notable exceptions: the X11 headers are full of declarations that ought to use `const` but do not; at one time, `libfreetype` also did not fully specify `const`, but this has not been checked recently.

**What must be avoided with const**

The GCC compiler warnings “assignment discards qualifiers...” and analogous warnings for “initialization”, “passing arg”, or “return” must be strenuously avoided in our own code. The only exception is when they are caused by X11 declarations or other third party code.

**What should be avoided with const**

A type cast, e.g., from `const char*` to `char*` does not solve any problems; depending on warning options, it may only hide them. Therefore such casts should be avoided whenever possible and otherwise must be carefully analyzed to make sure that they cannot cause the modification of quantities supposed to be constant.
9 Continuous integration

The TeX Live sources are subjected to continuous integration testing on Travis-CI (https://travis-ci.org/TeX-Live/texlive-source) via a git-svn mirror of the sources that is pushed to Github (https://github.com/TeX-Live/texlive-source). The git-svn mirror is updated (currently) at 30 minute intervals, and only the last commit pushed is tested on Travis-CI.

9.1 Transfer from Subversion to Github

The git-svn program (https://git-scm.com/docs/git-svn) is used to check out the sub-tree Build/source of the canonical Subversion repository. The author index file used is not maintained in either Git or Subversion but can be provided on request.

The initial checkout was done by invoking

```
git svn --authors-file usermap clone \
  svn://user@tug.org/texlive/trunk/Build/source
```

where the usermap file maps Subversion user names to name and emails of the authors. Anonymous checkout is also possible:

```
git svn --authors-file usermap clone \
  svn://tug.org/texlive/trunk/Build/source
```

In the following, we will use admin to refer to a user who has read/write access to the TeX Live subversion repository, and is also an administrator of the ‘TeX-Live’ team at Github. The above initial checkout has been carried out by admin on the server texlive.info.

On Github, a new git repository named texlive-source was created by admin within the TeX-Live “organization” (https://github.com/TeX-Live). The remote was added to the checkout with `git remote add origin git@github.com:TeX-Live/texlive-source.git`.

To automate the update on Github, a new ssh key was generated and added to the texlive-source repository on Github as deployment key. Thus, pushes using this key can only go to the texlive-source repository and not anywhere else.

The usage of git-svn requires a strict discipline to keep a linear history in the master branch. Since we are aiming at a pure mirror facility on Github, we have decided to further restrict the master branch of the texlive-source repository on Github to changes by admin.

This setup allows other developers to branch off master and push their branches to the Github repository, but all updates need to come from the local master (not the one on Github) to Subversion, back to master on texlive.info, and from there to Github.

9.2 Automatic update of the Git mirror

admin has installed a cron job on texlive.info running every 30 minute which essentially runs `git svn rebase` and `git push` in the master branch of the checkout. The first command fetches the changes from the Subversion repository and updates the master branch with them, and the second pushes changes (if any) to Github.
9.3 CI testing on Travis-CI

The source tree of TeX Live contains a top-level file .travis.yml which controls the automatic testing on Travis-CI. admin has registered with Travis-CI and allowed access to the Github’s TeX-Live organization’s texlive-source repository. The default settings are to build the last commit of each push. No further action is necessary on Travis-CI.

If changes have been pushed via the cron job above, Travis-CI will automatically checkout the last pushed commit and try building it.

9.4 Releases on Github

Given a git checkout of texlive-source:

```
git pull
git tag build-svnNNNN
git push --tags
```

and the result will appear at https://github.com/TeX-Live/texlive-source/releases. Releases can also be made manually from that web page (see tl-update-bindir for hints).
Appendix A install-tl

A.1 install-tl NAME
install-tl - TeX Live cross-platform installer

A.2 install-tl SYNOPSIS
install-tl [option]...
install-tl-windows.bat [option]...

A.3 install-tl DESCRIPTION
This installer creates a runnable TeX Live installation from various media, including over
the network, from local hard disk, a DVD, etc. The installer works on all platforms sup-
ported by TeX Live. For information on initially downloading TeX Live, see https://tug.
org/texlive/acquire.html.

The basic idea of TeX Live installation is for you to choose one of the top-level schemes,
each of which is defined as a different set of collections and packages, where a collection is
a set of packages, and a package is what contains actual files. Each package is in exactly
one collection, while schemes can contain any combination of packages and collections.

Within the installer, you can choose a scheme, and further customize the set of collections
to install, but not the set of the packages. To work at the package level, use tlmgr (reference
just below) after the initial installation is complete.

The default is scheme-full, which installs everything, and this is highly recommended.

A.4 REFERENCES
Post-installation configuration, package updates, and more, are handled through tlmgr(1),
the TeX Live Manager (https://tug.org/texlive/tlmgr.html).

The most up-to-date version of this installer documentation is on the Internet at

For the full documentation of TeX Live, see https://tug.org/texlive/doc.

A.5 install-tl EXAMPLES
With no options, install-tl drops you into an interactive menu where essentially all
default settings can be changed. With options, you can initialize the settings in various
ways, or perform the installation without interaction. Some examples:

install-tl --paper=letter
Initialize paper size setting. The only values allowed are letter and (the
default) a4.

install-tl --scheme scheme
Initialize the installation scheme; the default is full. For a list of schemes, see
the interactive $ menu.
install-tl --no-interaction
    Perform the installation immediately after parsing options, without entering
the interactive menu.

install-tl --profile texlive.profile
    Install, without interaction, according to the given TL profile file; see Sec-
section A.7 [PROFILES], page 41, below. To initialize from the profile and then
enter the interactive menu, add --init-from-profile.

Full documentation follows.

A.6 install-tl OPTIONS

As usual, all options can be specified in any order, and with either a leading - or --. An
argument value can be separated from its option by either a space or =.

The options relating to customization of the installation can also be selected in the
interactive installation menus (GUI or text).

-gui [|--module]
-no-gui
    If no module is given, starts the Tcl/Tk (see below) GUI installer.
    If module is given loads the given installer module. Currently the following
modules are supported:

    text
        The text mode user interface (default on Unix systems, including
Mac's). Same as the -no-gui option.

    tcl (or "perltk" or "wizard" or "expert" or nothing)
        The Tcl/Tk user interface (default on Windows). It starts with a
small number of configuration options, roughly equivalent to what
the former wizard option offers, but a button Advanced takes you
to a screen with roughly the same options as the former perltk
interface.

The default GUI requires Tcl/Tk. This was standard on Macs, but has been re-
moved in the latest macOS releases. It’s often already installed on GNU/Linux,
or can be easily installed through a distro package manager. For Windows, TeX
Live provides a Tcl/Tk runtime.

-lang llcode
    By default, the Tcl GUI uses the language detection built into Tcl/Tk. If that
fails you can select a different language by giving this option with a language
code (based on ISO 639-1). Currently supported (but not necessarily com-
pletely translated) are: English (en, default), Czech (cs), German (de), French
(fr), Italian (it), Japanese (ja), Dutch (nl), Polish (pl), Brazilian Portuguese
(pt_BR), Russian (ru), Slovak (sk), Slovenian (sl), Serbian (sr), Ukrainian (uk),
Vietnamese (vi), simplified Chinese (zh_CN), and traditional Chinese (zh_TW).

-repository url|path
    Specify the package repository to be used as the source of the installation. In
short, this can be a directory name or a url using http(s), ftp, or scp. The doc-
Appendix A: install-tl 37

umentation for \texttt{tlmgr} has the details (https://tug.org/texlive/doc/tlmgr.html#OPTIONS).

For installation, the default is to pick a mirror automatically, using https://mirror.ctan.org/systems/texlive/tlnet; the chosen mirror is then used for the entire download. You can use the special argument \texttt{ctan} as an abbreviation for this. (See https://ctan.org for more about CTAN and its mirrors.)

After installation is complete, you can use that installation as the repository for another installation. If you chose to install less than the full scheme containing all packages, the list of available schemes will be adjusted accordingly.

\texttt{-select-repository}

This option allows you to choose a particular mirror from the current list of active CTAN mirrors. This option is supported in the \texttt{text} and \texttt{gui} installer modes, and will also offer to install from local media if available, or from a repository specified on the command line. It’s useful when the (default) automatic redirection does not choose a good host for you.

\texttt{-all-options}

Normally options not relevant to the current platform are not shown (e.g., when running on Unix, Windows-specific options are omitted). Giving this command line option allows configuring such "foreign" settings.

\texttt{-custom-bin} \texttt{path}

If you have built your own set of TeX Live binaries (e.g., because precompiled binaries were not provided by TL for your platform), this option allows you to specify the \texttt{path} to a directory where the binaries for the current system are present. The installation will continue as usual, but at the end all files from \texttt{path} are copied over to \texttt{bin/custom/} under your installation directory and this \texttt{bin/custom/} directory is what will be added to the path for the post-install actions. To install multiple custom binary sets, manually rename \texttt{custom} before doing each.

For more information on custom binaries, see https://tug.org/texlive/custom-bin.html. For general information on building TeX Live, see https://tug.org/texlive/build.html.

\texttt{-debug-fakenet}

Pretend we’re doing a network install. This is for the sole purpose of testing the code to handle broken downloads, via moving package files aside in a tlnet mirror hierarchy.

\texttt{-debug-setup-vars}

Print final values of directory variables; for more debugging information on how they were set, also specify \texttt{-v}.

\texttt{-debug-translation}

In the former Perl/Tk GUI modes, this option reported any missing, or more likely untranslated, messages to standard error. Not yet implemented for the Tcl interface. Helpful for translators to see what remains to be done.
Instead of auto-detecting the current platform, use `platform`. Binaries for this platform must be present in `bin/platform/` and they must be runnable, or installation will fail. `-force-arch` is a synonym.

Display this help and exit. (This help is also on the web at https://tug.org/texlive/doc/install-tl.html). Sometimes the `perldoc` and/or `PAGER` programs on the system have problems, possibly resulting in control characters being literally output. This can’t always be detected, but you can set the `NOPERLDOC` environment variable and `perldoc` will not be used.

This is a quick-and-dirty installation option in case you already have an rsync or svn checkout of TeX Live. It will use the checkout as-is and will just do the necessary post-install. Be warned that the file `tlpkg/texlive.tlpdb` may be rewritten, that removal has to be done manually, and that the only realistic way to maintain this installation is to redo it from time to time. This option is not available via the installer interfaces. USE AT YOUR OWN RISK.

Similar to `-profile` (see Section A.7 [PROFILES], page 41, below), but only initializes the installation configuration from `profile_file` and then starts a normal interactive session. Environment variables are not ignored.

Write both all messages (informational, debugging, warnings) to `file`, in addition to standard output or standard error.

If this option is not given, the installer will create a log file in the root of the writable installation tree, for example, `/usr/local/texlive/YYYY/install-tl.log` for the `YYYY` release.

For the text mode installer only: do not clear the screen when entering a new menu. For debugging.

Quit early on installation failure of a non-core package.

By default, a few core packages are installed first; then, a failed installation of any other (non-core) package is noted, but does not stop the installation. Any such failed packages are retried, once.

If the retry also fails, by default the installer proceeds to completion anyway, with the idea that it was a transient network problem and reinstallation will succeed later. If this option is specified, and the retry fails, the installer aborts.

Do not install the documentation resp. source package files, both for the immediate installation and for future updates. After installation, inclusion of the `doc/src` files can be re-enabled via `tlmgr`:
Appendix A: install-tl

```
  tlmgr option docfiles 1
  tlmgr option srcfiles 1
```

If you later find that you want the doc/src files for a package that has been installed without them, you can get them like this (using the `fontspec` package as the example):

```
  tlmgr install --reinstall --with-doc --with-src fontspec
```

The source files mentioned here are those relating to TeX packages, such as `.dtx` files. The sources that are compiled to make the binaries are available separately: see https://tug.org/texlive/svn/.

-no-installation

Do not perform any installation. This is for debugging the initialization and setup routines without touching the disk.

-no-interaction

Do not enter the interactive menu; immediately perform the installation after initialization and option parsing. Also omit the check for a previous installation and asking about importing previous settings.

-no-persistent-downloads

Do not perform any installation. This is for debugging the initialization and setup routines without touching the disk.

-persistent-downloads

For network installs, activating this option makes the installer try to set up a persistent connection using the LWP Perl module. This opens only one connection between your computer and the server per session and reuses it, instead of initiating a new download for each package, which typically yields a significant speed-up.

This option is turned on by default, and the installation program will fall back to using `wget` if this is not possible. To disable usage of LWP and persistent connections, use -no-persistent-downloads.

-no-verify-downloads

By default, if a GnuPG `gpg` binary is found in PATH, downloads are verified against a cryptographic signature. This option disables such verification. The full description is in the Cryptographic Verification section of the `tlmgr` documentation, e.g., https://tug.org/texlive/doc/tlmgr.html#CRYPTOGRAPHIC-VERIFICATION

-non-admin

For Windows only: configure for the current user, not for all users.

-paper a4|letter

Set the default paper size for all TeX Live programs, as specified. The default is a4. The paper size can be set after installation with the `tlmgr paper` command.

-portable

Install for portable use, e.g., on a USB stick. See the `instopt_portable` description below for details.

-print-platform

Print the TeX Live identifier for the detected platform (hardware/operating system) combination to standard output, and exit. -print-arch is a synonym.
-profile profile_file

Load profile_file and do the installation with no user interaction, that is, a batch (unattended) install. Environment variables are ignored. See Section A.7 [PROFILES], page 41, below.

-q

Omit normal informational messages.

-scheme scheme

Schemes are the highest level of package grouping in TeX Live; the default is to use the full scheme, which includes everything. This option overrides that default. The scheme argument value may optionally have a prefix scheme-. The list of supported scheme names depends on what your package repository provides; see the interactive menu list.

-texdir dir

Specify the system installation directory; the default is /usr/local/texlive/YYYY for release YYYY. Specifying this option also causes the TEXMFLOCAL, TEXMFSYSCONFIG, and TEXMFSYSVAR directories to be set as subdirectories of dir, so they don’t have to be set individually.

There is a brief summary of these directories trees at Section A.9 [DIRECTORY TREES], page 44, below; for details on the trees set up by default, and their intended usage, see the main TeX Live documentation at https://tug.org/texlive/doc.

-texuserdir dir

Specify the user installation directory; the default is ~/.texliveYYYY (except on Macs, where there is no leading dot). Specifying this also causes the TEXMFHOME, TEXMFCONFIG, and TEXMFVAR directories to be set as subdirectories of dir.

-texmflocal dir

Specify the TEXMFLOCAL directory; the default is /usr/local/texlive/texmf-local, that is, one level up from the main installation. This is so locally-installed packages can be easily used across releases, which is usually desirable. Specifying the -texdir option changes this, putting TEXMFLOCAL under the main tree. The -texmflocal option can be used to specify an explicit directory. Anything installed here must follow the TeX directory structure (TDS), e.g.,TEXMFHOME/tex/latex/mypkg/mypkg.sty. TDS reference: https://tug.org/tds.

-texmfhome dir

Specify the TEXMFHOME directory; the default is ~/texmf, except on Macs, where it is ~/Library/texmf. Analogously to TEXMFLOCAL, the -texuserdir option changes this default.

Also as with TEXMFLOCAL, anything installed here must follow the TDS.

-texmfsyconfig dir

Specify the TEXMFSYSCONFIG and TEXMFSYSVAR system directories.
Appendix A: install-tl

-texmfconfig dir
-texmfvar dir

Specify the TEXMFCONFIG and TEXMFVAR user directories. The defaults are 
`.texliveYYYY/texmf-{config,var}`, except on Macs, where the leading dot 
is omitted (`/texliveYYYY/...`).

-v

Include verbose debugging messages; repeat for maximum debugging: -v -v. 
(Further repeats are accepted but ignored.)

-version, –version

Output version information and exit. If -v is also given, the versions of the 
TeX Live modules used are also reported.

A.7 PROFILES

A profile file normally contains all the values needed to perform an installation. After a 
normal installation has finished, a profile for that exact installation is written to the file 
tlpkg/texlive.profile. In addition, from the text menu one can select P to save the 
current setup as a profile at any time. These are small text files; feel free to peruse and edit 
them according to your needs.

Such a profile file can be given as the argument to -profile, for example to redo the 
each same installation on a different system. Alternatively, you can use a custom profile, 
most easily created by starting from a generated one and changing values. An empty profile 
file will cause the installer to use the defaults.

As mentioned above, the installer only supports selection by scheme and collections, not 
individual packages, so packages cannot be specified in profile files either. Use tlmgr to 
work at the package level.

Within a profile file, each line consists of

variable [value]

except for comment lines starting with #. The possible variable names are listed below. Values, when present, are either 0 or 1 for booleans, or strings (which must be specified 
without any quote characters). Leading whitespace is ignored.

If the variable selected_scheme is defined and no collection variables at all are defined, 
then the collections required by the specified scheme (which might change over time) are 
installed, without explicitly listing them. This eases maintenance of profile files. If any 
collections are specified in a profile, though, then the scheme is ignored and all desired 
collections must be given explicitly.

For example, a line

selected_scheme scheme-small

along with definitions for the installation directories (given below under "path options") 
suffices to install the "small" scheme with all default options. The schemes are described 
in the S menu in the text installer, or equivalent.

In addition to selected_scheme, here are the other variable names supported in a profile:

collection options (prefix collection-)
Collections are specified with a variable name with the prefix `collection-` followed by a collection name; there is no value. For instance, `collection-basic`. The collections are described in the C menu.

Schemes and collections (and packages) are ultimately defined by the files in the `tlpkg/tlpsrc/` source directory.

**path options**

It is best to define all of these, even though they may not be used in a given installation, so as to avoid unintentionally getting a default value that could cause problems later.

- `TEXDIR`
- `TEXMFLOCAL`
- `TEXMFSYSCONFIG`
- `TEXMFVAR`
- `TEXMFHOME`

**installer options** (prefix `instopt_`)

- `instopt_adjustpath` (default 0 on Unix, 1 on Windows)
  Adjust PATH environment variable.

- `instopt_adjustrepo` (default 1)
  Set remote repository to a multiplexed CTAN mirror after installation; see `-repository` above.

- `instopt_letter` (default 0)
  Set letter size paper as the default, instead of a4.

- `instopt_portable` (default 0)
  Install for portable use, e.g., on a USB stick, without touching the host system. Specifically, this forces the user directories `TEXMFHOME`, `TEXMFCONFIG`, `TEXMFVAR` to be identical to the system directories `TEXMFLOCAL`, `TEXMFSYSCONFIG`, `TEXMFVAR`, respectively (regardless of other options and environment variable.)

  In addition, on Windows, it disables the desktop integration, path adjustment, and file associations actions usually performed.

- `instopt_write18_restricted` (default 1)
  Enable \write18 for a restricted set of programs.

**tlpdb options** (prefix `tlpdbopt_`)

The definitive list is given in `tlpkg/TeXLive/TLConfig.pm`, in the hash `%TeXLive::TLConfig::TLDBOptions`, together with explanations. All items given there except for `tlpdbopt_location` can be specified. Here is the current list:

- `tlpdbopt_autobackup`
- `tlpdbopt_backupdir`
- `tlpdbopt_create_formats`
- `tlpdbopt_desktop_integration`
- `tlpdbopt_file_assocs`
platform options (prefix binary_)

For each supported platform in TeX Live (directories under bin/), the variable binary_PLATFORM can be set with value 1. For example:

binary_x86_64-linux 1

If no binary_ settings are made, the default is whatever the current machine is running.

In releases before 2017, many profile variables had different names (not documented here; see the install-tl source). They are accepted and transformed to the names given above. When a profile is written, the names above are always used.

For more details on all of the above options, consult the TeX Live installation manual, linked from https://tug.org/texlive/doc.

A.8 ENVIRONMENT VARIABLES

For ease in scripting and debugging, install-tl looks for the following environment variables. They are not of interest for normal user installations.

NOPERLDOC
Don’t try to run the --help message through perldoc.

TEXLIVE_DOWNLOADER
TL_DOWNLOAD_PROGRAM
TL_DOWNLOAD_ARGS

These override the normal choice of a download program; see the tlmgr documentation, e.g., https://tug.org/texlive/doc/tlmgr.html#ENVIRONMENT-VARIABLES.

TEXLIVE_INSTALL_ENV_NOCHECK
Omit the check for environment variables containing the string tex. People developing TeX-related software are likely to have many such variables.

TEXLIVE_INSTALL_NO_CONTEXT_CACHE
Omit creating the ConTeXt cache. This is useful for redistributors.

TEXLIVE_INSTALL_NO_DISKCHECK
If set to 1, omit free disk space check. By default, if a POSIX-compliant df program (supporting -Pk) is available, the installer checks for available disk space in the selected installation location, and will abort installation if there is insufficient disk space, plus a margin of 100MB. An equivalent check is made on Windows (not involving df).
**TEXLIVE_INSTALL_NO_RESUME**
Omit check for installing on top of a previous installation and then asking about importing previous settings.

**TEXLIVE_INSTALL_NO_WELCOME**
Omit printing the welcome message after successful installation, e.g., when testing.

**TEXLIVE_INSTALL_PAPER**
Set the default paper size for all relevant programs; must be either *letter* or *a4*. The default is *a4*.

**TEXLIVE_INSTALL_PREFIX**
**TEXLIVE_INSTALL_TEXMFCONFIG**
**TEXLIVE_INSTALL_TEXMFVAR**
**TEXLIVE_INSTALL_TEXMFHOME**
**TEXLIVE_INSTALL_TEXMFLOCAL**
**TEXLIVE_INSTALL_TEXMFSYSCONFIG**
**TEXLIVE_INSTALL_TEXMFSYSVAR**
Specify the respective directories. **TEXLIVE_INSTALL_PREFIX** defaults to `/usr/local/texlive`. All the defaults can be seen by running the installer interactively and then typing D for the directory menu.

The various command line options for specifying directories override these environment variables; since specifying both is usually accidental, a warning is given if the values are different.

### A.9 DIRECTORY TREES

There are a plethora of ways to specify the plethora of directory trees used by TeX Live. By far the simplest, and recommended, approach is not to change anything. The defaults suffice for the vast majority of installations.

But, for the sake of explanation, here is a table of the trees and the command line options that change them. The first group of three are system directories, and the second group of three are user directories; the two groups are quite analogous.

<table>
<thead>
<tr>
<th>tree</th>
<th>default</th>
<th>group change</th>
<th>single change</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>TEXMFLOCAL</strong></td>
<td>/usr/local/texlive/texmf-local</td>
<td>--texdir</td>
<td>--texmflocal</td>
</tr>
<tr>
<td><strong>TEXMFSYSVAR</strong></td>
<td>/usr/local/texlive/YYYY/texmf-var</td>
<td>--texdir</td>
<td>--texmfsysvar</td>
</tr>
<tr>
<td><strong>TEXMFSYSCONFIG</strong></td>
<td>/usr/local/texlive/YYYY/texmf-config</td>
<td>--texdir</td>
<td>--texmfsysconfig</td>
</tr>
<tr>
<td><strong>TEXMFHOME</strong></td>
<td>~/texmf</td>
<td>--texuserdir</td>
<td>--texmfhome</td>
</tr>
<tr>
<td><strong>TEXMFVAR</strong></td>
<td>~/texliveYYYY/texmf-var</td>
<td>--texuserdir</td>
<td>--texmfvar</td>
</tr>
<tr>
<td><strong>TEXMFCONFIG</strong></td>
<td>~/texliveYYYY/texmf-config</td>
<td>--texuserdir</td>
<td>--texmfconfig</td>
</tr>
</tbody>
</table>

In addition, as mentioned in the previous section, each tree has an environment variable **TEXLIVE_INSTALL_tree** which overrides the default; command line and profile settings both override environment variable settings.
The defaults vary slightly on Macs, as explained above in Section A.6 [OPTIONS], page 36.

For the user trees, the default value uses `~`, and this is left as a literal `~` in `texmf.cnf`. That way, each user can have their own `TEXMFHOME`, etc., as intended. On the other hand, for the system trees, if `~` is used during the installation, this is assumed to simply be a typing shorthand, and the expanded home directory is written in `texmf.cnf`, since it doesn’t make sense to have user-specific system directories.

For more on the directory trees and their intended usage, see the main TeX Live documentation at https://tug.org/texlive/doc.

A.10 install-tl BUGS
The `install-tl` script copies itself into the installed tree. Usually, it can be run from there, using the installed tree as the source for another installation. Occasionally, however, there may be incompatibilities in the code of the new `install-tl` and the infrastructure, resulting in (probably) inscrutable Perl errors. The way forward is to run `install-tl` out of the installer package (`install-tl-unx.tar.gz` or `install-tl.zip`) instead of the installation. Feel free to also report the issue; usually the code can be easily synced up again.

By the way, do not try to use `install-tl` to adjust options or installed packages in an existing installed tree. Use `tlmgr` instead.

A.11 AUTHORS AND COPYRIGHT
This script and its documentation were written for the TeX Live distribution (https://tug.org/texlive) and both are licensed under the GNU General Public License Version 2 or later.

$Id: install-tl 69711 2024-02-05 17:23:27Z karl $
Appendix B tlmgr

B.1 tlmgr NAME

tlmgr - the native TeX Live Manager

B.2 tlmgr SYNOPSIS

tlmgr [option...] action [option...] [operand...]

B.3 tlmgr DESCRIPTION

tlmgr manages an existing TeX Live installation, both packages and configuration options. For information on initially downloading and installing TeX Live, see https://tug.org/texlive/acquire.html.

The most up-to-date version of this documentation (updated nightly from the development sources) is available at https://tug.org/texlive/tlmgr.html, along with procedures for updating tlmgr itself and information about test versions.

TeX Live is organized into a few top-level schemes, each of which is specified as a different set of collections and packages, where a collection is a set of packages, and a package is what contains actual files. Schemes typically contain a mix of collections and packages, but each package is included in exactly one collection, no more and no less. A TeX Live installation can be customized and managed at any level.

See https://tug.org/texlive/doc for all the TeX Live documentation available.

B.4 tlmgr EXAMPLES

After successfully installing TeX Live, here are a few common operations with tlmgr:

   tlmgr option repository ctan

   tlmgr option repository https://mirror.ctan.org/systems/texlive/tlnet
   Tell tlmgr to use a nearby CTAN mirror for future updates; useful if you installed TeX Live from the DVD image and want to have continuing updates. The two commands are equivalent; ctan is just an alias for the given url.
   Caveat: mirror.ctan.org resolves to many different hosts, and they are not perfectly synchronized; we recommend updating only daily (at most), and not more often. You can choose a particular mirror if problems; the list of all CTAN mirrors with the status of each is at https://ctan.org/mirrors/mirmon.

   tlmgr update --list
   Report what would be updated without actually updating anything.

   tlmgr update --all
   Make your local TeX installation correspond to what is in the package repository (typically useful when updating from CTAN).

   tlmgr info what
   Display detailed information about a package what, such as the installation status and description, of searches for what in all packages.
tlmgr bug what

Display available bug-reporting information for what, a package or file name.

For all the capabilities and details of tlmgr, please read the following voluminous information.

B.5 tlmgr OPTIONS

The following options to tlmgr are global options, not specific to any action. All options, whether global or action-specific, can be given anywhere on the command line, and in any order. The first non-option argument will be the main action. In all cases, --option and -option are equivalent, and an = is optional between an option name and its value.

--repository url|path

Specify the package repository from which packages should be installed or updated, either a local directory or network location, as below. This overrides the default package repository found in the installation’s TeX Live Package Database (a.k.a. the TLPDB, which is given entirely in the file tlpkg/texlive.tlpdb).

This --repository option changes the location only for the current run; to make a permanent change, use option repository (see the Section B.6.16 [option], page 58, action).

As an example, you can choose a particular CTAN mirror with something like this:

--repository http://ctan.example.org/its/ctan/dir/systems/texlive/tlnet

Of course a real hostname and its particular top-level CTAN directory have to be specified. The list of CTAN mirrors is available at https://ctan.org/mirrors/mirmon.

Here’s an example of using a local directory:

--repository /local/TL/repository

For backward compatibility and convenience, --location and --repo are accepted as aliases for this option.

Locations can be specified as any of the following:

/some/local/dir
file:/some/local/dir

Equivalent ways of specifying a local directory.

c tan

https://mirror.ctan.org/systems/texlive/tlnet

Pick a CTAN mirror automatically, trying for one that is both nearby and up-to-date. The chosen mirror is used for the entire download. The bare ctan is merely an alias for the full url. (See https://ctan.org for more about CTAN and its mirrors.)

http://server/path/to/tlnet

Standard HTTP. If the (default) LWP method is used, persistent connections are supported. TL can also use curl or wget
to do the downloads, or an arbitrary user-specified program, as described in the tlmgr documentation (https://tug.org/texlive/doc/tlmgr.html#ENVIRONMENT-VARIABLES).

https://server/path/to/tlnet
Again, if the (default) LWP method is used, this supports persistent connections. Unfortunately, some versions of wget and curl do not support https, and even when wget supports https, certificates may be rejected even when the certificate is fine, due to a lack of local certificate roots. The simplest workaround for this problem is to use http or ftp.

ftp://server/path/to/tlnet
If the (default) LWP method is used, persistent connections are supported.

user@machine:/path/to/tlnet
scp://user@machine/path/to/tlnet
ssh://user@machine/path/to/tlnet

If the repository is on the network, trailing / characters and/or trailing /tlpkg and/or /archive components are ignored.

-gui [action]
Two notable GUI front-ends for tlmgr, tlshell and tlcockpit, are started up as separate programs; see their own documentation.

tlmgr itself has a graphical interface as well as the command line interface. You can give the option to invoke it, --gui, together with an action to be brought directly into the respective screen of the GUI. For example, running

```
tlmgr --gui update
```
starts you directly at the update screen. If no action is given, the GUI will be started at the main screen. See Section B.11 [GUI FOR TLMGR], page 74. However, the native GUI requires Perl/Tk, which is no longer included in TeX Live's Perl distribution for Windows. You may find tlshell or tlcockpit easier to work with.

-gui-lang llcode
By default, the GUI tries to deduce your language from the environment (on Windows via the registry, on Unix via LC_MESSAGES). If that fails you can select a different language by giving this option with a language code (based on ISO 639-1). Currently supported (but not necessarily completely translated) are: English (en, default), Czech (cs), German (de), French (fr), Italian (it), Japanese (ja), Dutch (nl), Polish (pl), Brazilian Portuguese (pt_BR), Russian (ru), Slovak (sk), Slovenian (sl), Serbian (sr), Ukrainian (uk), Vietnamese (vi), simplified Chinese (zh_CN), and traditional Chinese (zh_TW).

tlshell shares its message catalog with tlmgr.
Appendix B: \texttt{tlmgr} \hfill 49

\textbf{--command-logfile file}
\texttt{tlmgr} logs the output of all programs invoked (\texttt{mktexlr}, \texttt{mtxrun}, \texttt{fmtutil}, \texttt{updmap}) to a separate log file, by default \texttt{TEXMFSYSVAR/web2c/tlmgr-commands.log}. This option allows you to specify a different file for the log.

\textbf{--debug-translation}
In GUI mode, this switch tells \texttt{tlmgr} to report any untranslated (or missing) messages to standard error. This can help translators to see what remains to be done.

\textbf{--machine-readable}
Instead of the normal output intended for human consumption, write (to standard output) a fixed format more suitable for machine parsing. See the Section B.12 [MACHINE-READABLE OUTPUT], page 77, section below.

\textbf{--no-execute-actions}
Suppress the execution of the execute actions as defined in the \texttt{tlpsrc} files. Documented only for completeness, as this is only useful in debugging.

\textbf{--package-logfile file}
\texttt{tlmgr} logs all package actions (install, remove, update, failed updates, failed restores) to a separate log file, by default \texttt{TEXMFSYSVAR/web2c/tlmgr.log}. This option allows you to specify a different file for the log.

\textbf{--pause}
This option makes \texttt{tlmgr} wait for user input before exiting. Useful on Windows to avoid disappearing command windows.

\textbf{--persistent-downloads}
\textbf{--no-persistent-downloads}
For network-based installations, this option (on by default) makes \texttt{tlmgr} try to set up a persistent connection (using the \texttt{LWP} Perl module). The idea is to open and reuse only one connection per session between your computer and the server, instead of initiating a new download for each package.

If this is not possible, \texttt{tlmgr} will fall back to using \texttt{wget}. To disable these persistent connections, use \texttt{--no-persistent-downloads}.

\textbf{--pin-file}
Change the pinning file location from \texttt{TEXMFLOCAL/tlpkg/pinning.txt} (see Section B.10.1 [Pinning], page 73, below). Documented only for completeness, as this is only useful in debugging.

\textbf{--usermode}
Activates user mode for this run of \texttt{tlmgr}; see Section B.9 [USER MODE], page 71, below.

\textbf{--usertree} \texttt{dir}
Uses \texttt{dir} for the tree in user mode; see Section B.9 [USER MODE], page 71, below.
Appendix B: tlmgr

–verify-repo=[none|main|all]

Defines the level of verification done: If none is specified, no verification whatsoever is done. If main is given and a working GnuPG (gpg) binary is available, all repositories are checked, but only the main repository is required to be signed. If all is given, then all repositories need to be signed. See Section B.8 [CRYPTOGRAPHIC VERIFICATION], page 70, below for details.

The standard options for TeX Live programs are also accepted: --help/-h/-?, --version, -q (no informational messages), -v (debugging messages, can be repeated). For the details about these, see the TeXLive::TLUtils documentation.

The --version option shows version information about the TeX Live release and about the tlmgr script itself. If -v is also given, revision number for the loaded TeX Live Perl modules are shown, too.

B.6 ACTIONS

B.6.1 help

Display this help information and exit (same as --help, and on the web at https://tug.org/texlive/doc/tlmgr.html). Sometimes the perldoc and/or PAGER programs on the system have problems, resulting in control characters being literally output. This can’t always be detected, but you can set the NOPERLDOC environment variable and perldoc will not be used.

B.6.2 version

Gives version information (same as --version).

If -v has been given the revisions of the used modules are reported, too.

B.6.3 backup

backup [option...] –all
backup [option...] pkg...

If the --clean option is not specified, this action makes a backup of the given packages, or all packages given –all. These backups are saved to the value of the --backupdir option, if that is an existing and writable directory. If --backupdir is not given, the backupdir option setting in the TLPDB is used, if present. If both are missing, no backups are made. (The installer sets backupdir to .../tlpkg/backups, under the TL root installation directory, so it is usually defined; see the Section B.6.16 [option], page 58, description for more information.)

If the --clean option is specified, backups are pruned (removed) instead of saved. The optional integer value N may be specified to set the number of backups that will be retained when cleaning. If N is not given, the value of the autobackup option is used. If both are missing, an error is issued. For more details of backup pruning, see the option action.

Options:
Appendix B: tlmgr

---

**tlmgr**

- **backupdir** `directory`  
  Overrides the `backupdir` option setting in the TLPDB. The `directory` argument is required and must specify an existing, writable directory where backups are to be placed.

- **all**  
  If `--clean` is not specified, make a backup of all packages in the TeX Live installation; this will take quite a lot of space and time. If `--clean` is specified, all packages are pruned.

- **clean[=N]**  
  Instead of making backups, prune the backup directory of old backups, as explained above. The optional integer argument `N` overrides the `autobackup` option set in the TLPDB. You must use `--all` or a list of packages together with this option, as desired.

- **dry-run**  
  Nothing is actually backed up or removed; instead, the actions to be performed are written to the terminal.

---

### B.6.4 bug `[search-string]`

Searches for `search-string` (prompted for, if not given) as a package name and in package descriptions, as complete words, and in filenames, as any substring, and outputs bug-reporting and other information for the package selected from the results.

The search is equivalent to `tlmgr search --word --file search-string`. Thus, `search-string` is interpreted as a (Perl) regular expression.

---

### B.6.5 candidates pkg

Shows the available candidate repositories for package `pkg`. See Section B.10 [MULTIPLE REPOSITORIES], page 72, below.

---

### B.6.6 check `[option...]`

```
[depends|executes|files|runfiles|texmfdb|all]
```

Execute one (or all) check(s) of the consistency of the installation. If no problems are found, there will be no output. (To get a view of what is being done, run `tlmgr -v check`.)

- **depends**  
  Lists those packages which occur as dependencies in an installed collection, but are themselves not installed, and those packages which are not contained in any collection.

  If you call `tlmgr check collections` this test will be carried out instead since former versions for `tlmgr` called it that way.

- **executes**  
  Check that the files referred to by `execute` directives in the TeX Live Database are present.

- **files**
Appendix B: tlmgr

Checks that all files listed in the local TLPDB (\texttt{texlive.tlpdb}) are actually present, and lists those missing.

\textbf{runfiles}

List those filenames that are occurring more than one time in the runfiles sections, except for known duplicates.

\textbf{texmf dbs}

Checks related to the \texttt{ls-R} files. If you have defined new trees, or changed the \texttt{TEXMF} or \texttt{TEXMFDBS} variables, it can’t hurt to run this. It checks that:

- all items in \texttt{TEXMFDBS} have the \texttt{!!} prefix.
- all items in \texttt{TEXMFDBS} have an \texttt{ls-R} file (if they exist at all).
- all items in \texttt{TEXMF} with \texttt{!!} are listed in \texttt{TEXMFDBS}.
- all items in \texttt{TEXMF} with an \texttt{ls-R} file are listed in \texttt{TEXMFDBS}.

Options:

\texttt{--use-svn}

Use the output of \texttt{svn status} instead of listing the files; for checking the TL development repository. (This is run nightly.)

\textbf{B.6.7 conf}

\texttt{conf \{texmf|tlmgr|updmap\} \{--conffile \textit{file}\} \{--delete\} \{\textit{key} [\textit{value}]\}}

\texttt{conf auxtrees \{--conffile \textit{file}\} \{show|add|remove\} [\textit{value}]}

With only \texttt{conf}, show general configuration information for TeX Live, including active configuration files, path settings, and more. This is like running \texttt{texconfig conf}, but works on all supported platforms.

With one of \texttt{conf texmf}, \texttt{conf tlmgr}, or \texttt{conf updmap}, shows all key/value pairs (i.e., all settings) as saved in \texttt{ROOT/texmf.cnf}, the user-specific \texttt{tlmgr} configuration file (see below), or the first found (via \texttt{kpsewhich}) \texttt{updmap.cfg} file, respectively.

If \textit{key} is given in addition, shows the value of only that \textit{key} in the respective file. If option \texttt{--delete} is also given, the value in the given configuration file is entirely removed (not just commented out).

If \textit{value} is given in addition, \textit{key} is set to \textit{value} in the respective file. \textit{No error checking is done!}

The \texttt{PATH} value shown by \texttt{conf} is as used by \texttt{tlmgr}. The directory in which the \texttt{tlmgr} executable is found is automatically prepended to the \texttt{PATH} value inherited from the environment.

Here is a practical example of changing configuration values. If the execution of (some or all) system commands via \texttt{\write18} was left enabled during installation, you can disable it afterwards:

\texttt{tlmgr conf texmf shell_escape 0}

The subcommand \texttt{auxtrees} allows adding and removing arbitrary additional texmf trees, completely under user control. \texttt{auxtrees show} shows the list of additional trees, \texttt{auxtrees add tree} adds a tree to the list, and \texttt{auxtrees}
remove tree removes a tree from the list (if present). The trees should not contain an ls-R file (or files will not be found if the ls-R becomes stale). This works by manipulating the Kpathsea variable TExMFAUXTREES, in (by default) ROOT/texmf.cnf. Example:

tlmgr conf auxtrees add /quick/test/tree
   tlmgr conf auxtrees remove /quick/test/tree

In all cases the configuration file can be explicitly specified via the option --conffile file, e.g., if you don't want to change the system-wide configuration.

Warning: The general facility for changing configuration values is here, but tinkering with settings in this way is strongly discouraged. Again, no error checking on either keys or values is done, so any sort of breakage is possible.

B.6.8 dump-tlpdb [option...] [--json]

Dump complete local or remote TLPDB to standard output, as-is. The output is analogous to the --machine-readable output; see Section B.12 [MACHINE-READABLE OUTPUT], page 77, section.

Options:

-local
   Dump the local TLPDB.

-remote
   Dump the remote TLPDB.

-json
   Instead of dumping the actual content, the database is dumped as JSON. For the format of JSON output see tlpkg/doc/JSON-formats.txt, format definition TLPDB.

Exactly one of --local and --remote must be given.

In either case, the first line of the output specifies the repository location, in this format:
"location-url" "\t" location

where location-url is the literal field name, followed by a tab, and location is the file or url to the repository.

Line endings may be either LF or CRLF depending on the current platform.

B.6.9 generate

generate [option...] language
generate [option...] language.dat
generate [option...] language.def
generate [option...] language.dat.lua

The generate action overwrites any manual changes made in the respective files: it recreates them from scratch based on the information of the installed packages, plus local adaptions. The TeX Live installer and tlmgr routinely call generate for all of these files.
For managing your own fonts, please read the `updmap --help` information and/or https://tug.org/fonts/fontinstall.html.

For managing your own formats, please read the `fmtutil --help` information.

In more detail: `generate` remakes any of the configuration files `language.dat`, `language.def`, and `language.dat.lua` from the information present in the local TLPDB, plus locally-maintained files.

The locally-maintained files are `language-local.dat`, `language-local.def`, or `language-local.dat.lua`, searched for in `TEXMFLOCAL` in the respective directories. If local additions are present, the final file is made by starting with the main file, omitting any entries that the local file specifies to be disabled, and finally appending the local file.

(Historical note: The formerly supported `updmap-local.cfg` and `fmtutil-local.cnf` are no longer read, since `updmap` and `fmtutil` now reads and supports multiple configuration files. Thus, local additions can and should be put into an `updmap.cfg` of `fmtutil.cnf` file in `TEXMFLOCAL`. The `generate updmap` and `generate fmtutil` actions no longer exist.)

Local files specify entries to be disabled with a comment line, namely one of these:

```
%!NAME
--!NAME
```

where `language.dat` and `language.def` use `%`, and `language.dat.lua` use `--`. In all cases, the `name` is the respective format name or hyphenation pattern identifier. Examples:

```
%!german
--!usenglishmax
```

(Of course, you’re not likely to actually want to disable those particular items. They’re just examples.)

After such a disabling line, the local file can include another entry for the same item, if a different definition is desired. In general, except for the special disabling lines, the local files follow the same syntax as the master files.

The form `generate language` recreates all three files `language.dat`, `language.def`, and `language.dat.lua`, while the forms with an extension recreates only that given language file.

Options:

```
--dest output_file
```

specifies the output file (defaults to the respective location in `TEXMFSYSVAR`). If `--dest` is given to `generate language`, it serves as a basename onto which `.dat` will be appended for the name of the `language.dat` output file, `.def` will be appended to the value for the name of the `language.def` output file, and `.dat.lua` to the name of the `language.dat.lua` file. (This is just to avoid overwriting; if you want a specific name for each output file, we recommend invoking `tlmgr` twice.)

```
--localcfg local_conf_file
```

specifies the (optional) local additions (defaults to the respective location in `TEXMFLOCAL`).

```
--rebuild-sys
```

tells `tlmgr` to run necessary programs after config files have been regenerated. These are: `fmtutil-sys --all` after `generate fmtutil`, `fmtutil-sys`...
--byhyphen .../language.dat after generate language.dat, and fmtutil-
sys --byhyphen .../language.def after generate language.def.
These subsequent calls cause the newly-generated files to actually take effect. This is not done by default since those calls are lengthy processes and one might want to made several related changes in succession before invoking these programs.

The respective locations are as follows:
tex/generic/config/language.dat (and language-local.dat)
tex/generic/config/language.def (and language-local.def)
tex/generic/config/language.dat.lua (and language-local.dat.lua)

B.6.10 gui
Start the graphical user interface. See GUI below.

B.6.11 info

info [option...] pkg...
info [option...] collections
info [option...] schemes

With no argument, lists all packages available at the package repository, prefixing those already installed with i.

With the single word collections or schemes as the argument, lists the request type instead of all packages.

With any other arguments, display information about pkg: the name, category, short and long description, sizes, installation status, and TeX Live revision number. If pkg is not locally installed, searches in the remote installation source.

For normal packages (not collections or schemes), the sizes of the four groups of files (run/src/doc/bin files) are shown separately. For collections, the cumulative size is shown, including all directly-dependent packages (but not dependent collections). For schemes, the cumulative size is also shown, including all directly-dependent collections and packages.

If pkg is not found locally or remotely, the search action is used and lists matching packages and files.

It also displays information taken from the TeX Catalogue, namely the package version, date, and license. Consider these, especially the package version, as approximations only, due to timing skew of the updates of the different pieces. By contrast, the revision value comes directly from TL and is reliable.

The former actions show and list are merged into this action, but are still supported for backward compatibility.

Options:

--list

If the option --list is given with a package, the list of contained files is also shown, including those for platform-specific dependencies. When given with schemes and collections, --list outputs their dependencies in a similar way.
**–only-installed**
If this option is given, the installation source will not be used; only locally installed packages, collections, or schemes are listed.

**–only-remote**
Only list packages from the remote repository. Useful when checking what is available in a remote repository using `tlmgr --repo ... --only-remote info`. Note that `--only-installed` and `--only-remote` cannot both be specified.

**–data item1,item2,...**
If the option `--data` is given, its argument must be a comma or colon separated list of field names from: name, category, localrev, remoterev, shortdesc, longdesc, installed, size, relocatable, depends, cat-version, cat-date, cat-license, plus various cat-contact-* fields (see below).

The cat-* fields all come from the TeX Catalogue ([https://ctan.org/pkg/catalogue](https://ctan.org/pkg/catalogue)). For each, there are two more variants with prefix l and r, e.g., lcat-version and rcat-version, which indicate the local and remote information, respectively. The variants without l and r show the most current one, which is normally the remote value.

The requested packages’ information is listed in CSV format, one package per line, and the column information is given by the itemN. The depends column contains the names of all the dependencies separated by : characters.

At this writing, the cat-contact-* fields include: home, repository, support, bugs, announce, development. Each may be empty or a url value. A brief description is on the CTAN upload page for new packages: [https://ctan.org/upload](https://ctan.org/upload).

**–json**
In case `--json` is specified, the output is a JSON encoded array where each array element is the JSON representation of a single TLPOBJ but with additional information. For details see `tlpkg/doc/JSON-formats.txt`, format definition: TLPOBJINFO. If both `--json` and `--data` are given, `--json` takes precedence.

**B.6.12 init-usertree**
Sets up a texmf tree for so-called user mode management, either the default user tree (TEXMFHOME), or one specified on the command line with `--usertree`. See Section B.9 [USER MODE], page 71, below.

**B.6.13 install [option...] pkg...**
Install each pkg given on the command line, if it is not already installed. It does not touch existing packages; see the update action for how to get the latest version of a package.

By default this also installs all packages on which the given pkgs are dependent. Options: `–dry-run`
Appendix B: tlmgr

Nothing is actually installed; instead, the actions to be performed are written to the terminal.

-file

Instead of fetching a package from the installation repository, use the package files given on the command line. These files must be standard TeX Live package files (with contained tlpobj file).

-force

If updates to tlmgr itself (or other parts of the basic infrastructure) are present, tlmgr will bail out and not perform the installation unless this option is given. Not recommended.

-no-depends

Do not install dependencies. (By default, installing a package ensures that all dependencies of this package are fulfilled.)

-no-depends-at-all

Normally, when you install a package which ships binary files the respective binary package will also be installed. That is, for a package foo, the package foo.i386-linux will also be installed on an i386-linux system. This option suppresses this behavior, and also implies --no-depends. Don’t use it unless you are sure of what you are doing.

-reinstall

Reinstall a package (including dependencies for collections) even if it already seems to be installed (i.e, is present in the TLPDB). This is useful to recover from accidental removal of files in the hierarchy.

When re-installing, only dependencies on normal packages are followed (i.e., not those of category Scheme or Collection).

-with-doc

-with-src

While not recommended, the install-tl program provides an option to omit installation of all documentation and/or source files. (By default, everything is installed.) After such an installation, you may find that you want the documentation or source files for a given package after all. You can get them by using these options in conjunction with --reinstall, as in (using the fontspec package as the example):

```
tlmgr install --reinstall --with-doc --with-src fontspec
```

This action does not automatically add new symlinks in system directories; you need to run tlmgr path add (Section B.6.18 [path], page 60) yourself if you are using this feature and want new symlinks added.

B.6.14 key

key list
Appendix B: tlmgr

**key add file**

**key remove keyid**

The action `key` allows listing, adding and removing additional GPG keys to the set of trusted keys, that is, those that are used to verify the TeX Live databases.

With the `list` argument, `key` lists all keys.

The `add` argument requires another argument, either a filename or `-` for stdin, from which the key is added. The key is added to the local keyring `GNUPGHOME/repository-keys.gpg`, which is normally `tlpkg/gpg/repository-keys.gpg`.

The `remove` argument requires a key id and removes the requested id from the local keyring.

**B.6.15 list**

Synonym for Section B.6.11 [info], page 55.

**B.6.16 option**

```bash
option [-json] [show]
option [-json] showall|help
option key [value]
```

The first form, `show`, shows the global TeX Live settings currently saved in the TLPDB with a short description and the key used for changing it in parentheses.

The second form, `showall`, is similar, but also shows options which can be defined but are not currently set to any value (`help` is a synonym).

Both `show...` forms take an option `--json`, which dumps the option information in JSON format. In this case, both forms dump the same data. For the format of the JSON output see `tlpkg/doc/JSON-formats.txt`, format definition `TLOPTION`.

In the third form, with `key`, if `value` is not given, the setting for `key` is displayed. If `value` is present, `key` is set to `value`.

Possible values for `key` are (run `tlmgr option showall` for the definitive list):

- `repository` (default package repository),
- `formats` (generate formats at installation or update time),
- `postcode` (run postinst code blobs),
- `docfiles` (install documentation files),
- `srcfiles` (install source files),
- `backupdir` (default directory for backups),
- `autobackup` (number of backups to keep),
- `sys_bin` (directory to which executables are linked by the path action),
- `sys_man` (directory to which man pages are linked by the path action),
- `sys_info` (directory to which Info files are linked by the path action),
- `desktop_integration` (Windows-only: create Start menu shortcuts),
- `fileassocs` (Windows-only: change file associations),
- `multiuser` (Windows-only: install for all users)

One common use of `option` is to permanently change the installation to get further updates from the Internet, after originally installing from DVD. To do this, you can run `tlmgr option repository https://mirror.ctan.org/systems/texlive/tlnet`
Appendix B: tlmgr

The `install-tl` documentation has more information about the possible values for `repository`. (For backward compatibility, `location` can be used as a synonym for `repository`.)

If `formats` is set (this is the default), then formats are regenerated when either the engine or the format files have changed. Disable this only when you know how and want to regenerate formats yourself whenever needed (which is often, in practice).

The `postcode` option controls execution of per-package postinstallation action code. It is set by default, and again disabling is not likely to be of interest except to developers doing debugging.

The `docfiles` and `srcfiles` options control the installation of their respective file groups (documentation, sources; grouping is approximate) per package. By default both are enabled (1). Either or both can be disabled (set to 0) if disk space is limited or for minimal testing installations, etc. When disabled, the respective files are not downloaded at all.

The options `autobackup` and `backupdir` determine the defaults for the actions `update`, `backup` and `restore`. These three actions need a directory in which to read or write the backups. If `--backupdir` is not specified on the command line, the `backupdir` option value is used (if set). The TL installer sets `backupdir` to `.../tlpkg/backups`, under the TL root installation directory.

The `autobackup` option (de)activates automatic generation of backups. Its value is an integer. If the `autobackup` value is -1, no backups are removed. If `autobackup` is 0 or more, it specifies the number of backups to keep. Thus, backups are disabled if the value is 0. In the `--clean` mode of the `backup` action this option also specifies the number to be kept. The default value is 1, so that backups are made, but only one backup is kept.

To setup `autobackup` to -1 on the command line, use:

```
tlmgr option -- autobackup -1
```

The `--` avoids having the `-1` treated as an option. (The `--` stops parsing for options at the point where it appears; this is a general feature across most Unix programs.)

The `sys_bin`, `sys_man`, and `sys_info` options are used on Unix systems to control the generation of links for executables, Info files and man pages. See the `path` action for details.

The last three options affect behavior on Windows installations. If `desktop_integration` is set, then some packages will install items in a sub-folder of the Start menu for `tlmgr gui`, documentation, etc. If `fileassocs` is set, Windows file associations are made (see also the `postaction` action). Finally, if `multiuser` is set, then adaptions to the registry and the menus are done for all users on the system instead of only the current user. All three options are on by default.

### B.6.17 paper

```
paper [a4|letter]
<|xdvi|pdftex|dvips|dvipdfmx|context|psutils] paper [papersize|--list]
paper -json
```

With no arguments (`tlmgr paper`), shows the default paper size setting for all known programs.
With one argument (e.g., `tlmgr paper a4`), sets the default for all known programs to that paper size.

With a program given as the first argument and no paper size specified (e.g., `tlmgr dvips paper`), shows the default paper size for that program.

With a program given as the first argument and a paper size as the last argument (e.g., `tlmgr dvips paper a4`), set the default for that program to that paper size.

With a program given as the first argument and `--list` given as the last argument (e.g., `tlmgr dvips paper --list`), shows all valid paper sizes for that program. The first size shown is the default.

If `--json` is specified without other options, the paper setup is dumped in JSON format. For the format of JSON output see `tlpkg/doc/JSON-formats.txt`, format definition `TLPAPER`.

Incidentally, this syntax of having a specific program name before the `paper` keyword is unusual. It is inherited from the longstanding `texconfig` script, which supports other configuration settings for some programs, notably `dvips`. `tlmgr` does not support those extra settings.

**B.6.18 path**

```
path [-windowsmode=user|admin] add
path [-windowsmode=user|admin] remove
```

On Unix, adds or removes symlinks for executables, man pages, and info pages in the system directories specified by the respective options (see the Section B.6.16 [option], page 58, description above). Does not change any initialization files, either system or personal. Furthermore, any executables added or removed by future updates are not taken care of automatically; this command must be rerun as needed.

On Windows, the registry part where the binary directory is added or removed is determined in the following way:

If the user has admin rights, and the option `--windowsmode` is not given, the setting `w32_multi_user` determines the location (i.e., if it is on then the system path, otherwise the user path is changed).

If the user has admin rights, and the option `--windowsmode` is given, this option determines the path to be adjusted.

If the user does not have admin rights, and the option `--windowsmode` is not given, and the setting `w32_multi_user` is off, the user path is changed, while if the setting `w32_multi_user` is on, a warning is issued that the caller does not have enough privileges.

If the user does not have admin rights, and the option `--windowsmode` is given, it must be `user` and the user path will be adjusted. If a user without admin rights uses the option `--windowsmode admin` a warning is issued that the caller does not have enough privileges.

**B.6.19 pinning**

The `pinning` action manages the pinning file, see Section B.10.1 [Pinning], page 73, below.
pinning show
   Shows the current pinning data.

pinning add repo pkgglob...
   Pins the packages matching the pkgglob(s) to the repository repo.

pinning remove repo pkgglob...
   Any packages recorded in the pinning file matching the <pkgglob>s for the given repository repo are removed.

pinning remove repo --all
   Remove all pinning data for repository repo.

B.6.20 platform

platform list | add | remove platform...
platform set platform
platform set auto
   platform list lists the TeX Live names of all the platforms (a.k.a. architectures), (i386-linux, ...) available at the package repository.
   platform add platform... adds the executables for each given platform platform to the installation from the repository.
   platform remove platform... removes the executables for each given platform platform from the installation, but keeps the currently running platform in any case.
   platform set platform switches TeX Live to always use the given platform instead of auto detection.
   platform set auto switches TeX Live to auto detection mode for platform.
   Platform detection is needed to select the proper xz and wget binaries that are shipped with TeX Live.
   arch is a synonym for platform.
Options:
   --dry-run
      Nothing is actually installed; instead, the actions to be performed are written to the terminal.

B.6.21 postaction

postaction [option...] install [shortcut | fileassoc | script] [pkg...]
postaction [option...] remove [shortcut | fileassoc | script] [pkg...]
   Carry out the postaction shortcut, fileassoc, or script given as the second required argument in install or remove mode (which is the first required argument), for either the packages given on the command line, or for all if --all is given.
Options:
   --windowsmode=[user | admin]
      If the option --windowsmode is given the value user, all actions will only be carried out in the user-accessible parts of the
Appendix B: tlmgr

registry/filesystem, while the value admin selects the system-wide parts of the registry for the file associations. If you do not have enough permissions, using --windowsmode=admin will not succeed.

--fileassocmode=[1|2]
--fileassocmode specifies the action for file associations. If it is set to 1 (the default), only new associations are added; if it is set to 2, all associations are set to the TeX Live programs. (See also option fileassocs.)

--all

Carry out the postactions for all packages

B.6.22 print-platform
Print the TeX Live identifier for the detected platform (hardware/operating system) combination to standard output, and exit. --print-arch is a synonym.

B.6.23 print-platform-info
Print the TeX Live platform identifier, TL platform long name, and original output from guess.

B.6.24 remove [option...] pkg...
Remove each pkg specified. Removing a collection removes all package dependencies (unless --no-depends is specified), but not any collection dependencies of that collection. However, when removing a package, dependencies are never removed. Options:

--all
Uninstalls all of TeX Live, asking for confirmation unless --force is also specified.

--backup

--backupdir directory
These options behave just as with the Section B.6.31 [update], page 66, action (q.v.), except they apply to making backups of packages before they are removed. The default is to make such a backup, that is, to save a copy of packages before removal.

The Section B.6.26 [restore], page 64, action explains how to restore from a backup.

--no-depends
Do not remove dependent packages.

--no-depends-at-all
See above under Section B.6.13 [install], page 56, (and beware).

--force
By default, removal of a package or collection that is a dependency of another collection or scheme is not allowed. With this option, the package will be removed unconditionally. Use with care.
A package that has been removed using the --force option because it is still listed in an installed collection or scheme will not be updated, and will be mentioned as forcibly removed in the output of tlmgr update --list.

--dry-run

Nothing is actually removed; instead, the actions to be performed are written to the terminal.

Except with --all, this remove action does not automatically remove symlinks to executables from system directories; you need to run tlmgr path remove (Section B.6.18 [path], page 60) yourself if you remove an individual package with a symlink in a system directory.

B.6.25 repository

repository list
repository list path|url|tag
repository add path [tag]
repository remove path|tag
repository set path[#tag] [path[#tag] ...]
repository status

This action manages the list of repositories. See MULTIPLE_REPOSITORIES below for detailed explanations.

The first form, repository list, lists all configured repositories and the respective tags if set. If a path, url, or tag is given after the list keyword, it is interpreted as the source from which to initialize a TL database and lists the contained packages. This can also be an otherwise-unused repository, either local or remote. If the option --with-platforms is specified in addition, for each package the available platforms (if any) are also listed.

The form repository add adds a repository (optionally attaching a tag) to the list of repositories, while repository remove removes a repository, either by full path/url, or by tag.

The form repository set sets the list of available repositories to the items given on the command line, overwriting previous settings.

The form repository status reports the verification status of the loaded repositories with the format of one repository per line with fields separated by a single space:

The tag (which can be the same as the url);

= the url;

= iff machine-readable output is specified, the verification code (a number);

= a textual description of the verification status, as the last field extending to the end of line.

That is, in normal (not machine-readable) output, the third field (numeric verification status) is not present.

In all cases, one of the repositories must be tagged as main; otherwise, all operations will fail!
B.6.26 restore

restore [option...] pkg [rev]
restore [option...] --all

Restore a package from a previously-made backup.
If --all is given, try to restore the latest revision of all package backups found in the backup directory.
Otherwise, if neither pkg nor rev are given, list the available backup revisions for all packages. With pkg given but no rev, list all available backup revisions of pkg.
When listing available packages, tlmgr shows the revision, and in parenthesis the creation time if available (in format yyyy-mm-dd hh:mm).
If (and only if) both pkg and a valid revision number rev are specified, try to restore the package from the specified backup.
Options:

--all
Try to restore the latest revision of all package backups found in the backup directory. Additional non-option arguments (like pkg) are not allowed.

--backupdir directory
Specify the directory where the backups are to be found. If not given it will be taken from the configuration setting in the TLPDB.

--dry-run
Nothing is actually restored; instead, the actions to be performed are written to the terminal.

--force
Don’t ask questions.

--json
When listing backups, the option --json turn on JSON output. The format is an array of JSON objects (name, rev, date). For details see tlpkg/doc/JSON-formats.txt, format definition: TLBACKUPS. If both --json and --data are given, --json takes precedence.

B.6.27 search

search [option...] what
search [option...] --file what
search [option...] --all what

By default, search the names, short descriptions, and long descriptions of all locally installed packages for the argument what, interpreted as a (Perl) regular expression.
Options:

--file
List all filenames containing *what*.

---

Search everything: package names, descriptions and filenames.

---

Search the TeX Live Database of the installation medium, instead of the local installation.

---

Restrict the search of package names and descriptions (but not filenames) to match only full words. For example, searching for *table* with this option will not output packages containing the word *tables* (unless they also contain the word *table* on its own).

### B.6.28 shell

Starts an interactive mode, where tlmgr prompts for commands. This can be used directly, or for scripting. The first line of output is `protocol n`, where *n* is an unsigned number identifying the protocol version (currently 1).

In general, tlmgr actions that can be given on the command line translate to commands in this shell mode. For example, you can say `update --list` to see what would be updated. The TLPDB is loaded the first time it is needed (not at the beginning), and used for the rest of the session.

Besides these actions, a few commands are specific to shell mode:

- `protocol`
  
  Print `protocol n`, the current protocol version.

- `help`
  
  Print pointers to this documentation.

- `version`
  
  Print tlmgr version information.

- `quit`, `end`, `bye`, `byebye`, `EOF`
  
  Exit.

- `restart`
  
  Restart `tlmgr shell` with the original command line; most useful when developing `tlmgr`.

- `load [local|remote]`
  
  Explicitly load the local or remote, respectively, TLPDB.

- `save`
  
  Save the local TLPDB, presumably after other operations have changed it.

- `get [var] =item set [var [val]]`
  
  Get the value of *var*, or set it to *val*. Possible *var* names: `debug-translation`, `machine-readable`, `no-execute-actions`, `require-verification`, `verify-downloads`, `repository`, and `prompt`. All except `repository` and `prompt` are
booleans, taking values 0 and 1, and behave like the corresponding command line option. The repository variable takes a string, and sets the remote repository location. The prompt variable takes a string, and sets the current default prompt.

If var or then val is not specified, it is prompted for.

B.6.29 show
Synonym for Section B.6.11 [info], page 55.

B.6.30 uninstall
Synonym for Section B.6.24 [remove], page 62.

B.6.31 update [option...] [pkg...]
Updates the packages given as arguments to the latest version available at the installation source. Either --all or at least one pkg name must be specified. Options:

--all
Update all installed packages except for tlmgr itself. If updates to tlmgr itself are present, this gives an error, unless also the option --force or --self is given. (See below.)

In addition to updating the installed packages, during the update of a collection the local installation is (by default) synchronized to the status of the collection on the server, for both additions and removals.

This means that if a package has been removed on the server (and thus has also been removed from the respective collection), tlmgr will remove the package in the local installation. This is called “auto-remove” and is announced as such when using the option --list. This auto-removal can be suppressed using the option --no-auto-remove (not recommended, see option description).

Analogously, if a package has been added to a collection on the server that is also installed locally, it will be added to the local installation. This is called “auto-install” and is announced as such when using the option --list. This auto-installation can be suppressed using the option --no-auto-install (also not recommended).

An exception to the collection dependency checks (including the auto-installation of packages just mentioned) are those that have been “forcibly removed” by you, that is, you called tlmgr remove --force on them. (See the remove action documentation.) To reinstall any such forcibly removed packages use --reinstall-forcibly-removed.

To reiterate: automatic removals and additions are entirely determined by comparison of collections. Thus, if you manually install an individual package foo which is later removed from the server, tlmgr will not notice and will not remove it locally. (It has to be this way, without major rearchitecture work, because the tlpdb does not record the repository from which packages come from.)

If you want to exclude some packages from the current update run (e.g., due to a slow link), see the --exclude option below.
Appendix B: tlmgr

–self
Update tlmgr itself (that is, the infrastructure packages) if updates to it are present. On Windows this includes updates to the private Perl interpreter shipped inside TeX Live.

If this option is given together with either --all or a list of packages, then tlmgr will be updated first and, if this update succeeds, the new version will be restarted to complete the rest of the updates.

In short:

```
\texttt{tlmgr update --self} \quad \# \text{update infrastructure only}
\texttt{tlmgr update --self --all} \quad \# \text{update infrastructure and all packages}
\texttt{tlmgr update --force --all} \quad \# \text{update all packages but *not* infrastructure}
\quad \# \ldots \text{this last at your own risk, not recommended!}
```

–dry-run
Nothing is actually installed; instead, the actions to be performed are written to the terminal. This is a more detailed report than --list.

–list [pkg]
Concisely list the packages which would be updated, newly installed, or removed, without actually changing anything. If --all is also given, all available updates are listed. If --self is given, but not --all, only updates to the critical packages (tlmgr, texlive infrastructure, perl on Windows, etc.) are listed. If neither --all nor --self is given, and in addition no pkg is given, then --all is assumed (thus, \texttt{tlmgr update --list} is the same as \texttt{tlmgr update --list --all}). If neither --all nor --self is given, but specific package names are given, those packages are checked for updates.

–exclude pkg
Exclude pkg from the update process. If this option is given more than once, its arguments accumulate.

An argument pkg excludes both the package pkg itself and all its related platform-specific packages pkg.ARCH. For example,

```
\texttt{tlmgr update --all --exclude a2ping}
```

will not update a2ping, a2ping.1386-linux, or any other a2ping.ARCH package.

If this option specifies a package that would otherwise be a candidate for auto-installation, auto-removal, or reinstallation of a forcibly removed package, tlmgr quits with an error message. Excludes are not supported in these circumstances.

This option can also be set permanently in the tlmgr config file with the key update-exclude.

–no-auto-remove [pkg...]
By default, tlmgr tries to remove packages in an existing collection which have disappeared on the server, as described above under --all. This option prevents such removals, either for all packages (with --all), or for just the given pkg names. This can lead to an inconsistent TeX installation, since packages
are not infrequently renamed or replaced by their authors. Therefore this is not recommended.

-no-auto-install [pkg...]
Under normal circumstances \texttt{tlmgr} will install packages which are new on the server, as described above under \texttt{--all}. This option prevents any such automatic installation, either for all packages (with \texttt{--all}), or the given \texttt{pkg} names.

Furthermore, after the \texttt{tlmgr} run using this has finished, the packages that would have been auto-installed will be considered as forcibly removed. So, if \texttt{foobar} is the only new package on the server, then

\begin{verbatim}
  tlmgr update --all --no-auto-install
\end{verbatim}
is equivalent to

\begin{verbatim}
  tlmgr update --all
  tlmgr remove --force foobar
\end{verbatim}

Again, since packages are sometimes renamed or replaced, using this option is not recommended.

-reinstall-forcibly-removed
Under normal circumstances \texttt{tlmgr} will not install packages that have been forcibly removed by the user; that is, removed with \texttt{remove --force}, or whose installation was prohibited by \texttt{--no-auto-install} during an earlier update.

This option makes \texttt{tlmgr} ignore the forcible removals and re-install all such packages. This can be used to completely synchronize an installation with the server’s idea of what is available:

\begin{verbatim}
  tlmgr update --reinstall-forcibly-removed --all
\end{verbatim}

-backup
-backupdir directory
These two options control the creation of backups of packages before updating; that is, backing up packages as currently installed. If neither option is given, no backup will made. If \texttt{--backupdir} is given and specifies a writable directory then a backup will be made in that location. If only \texttt{--backup} is given, then a backup will be made to the directory previously set via the Section B.6.16 [option], page 58, action (see below). If both are given then a backup will be made to the specified directory.

You can also set options via the Section B.6.16 [option], page 58, action to automatically make backups for all packages, and/or keep only a certain number of backups.

\texttt{tlmgr} always makes a temporary backup when updating packages, in case of download or other failure during an update. In contrast, the purpose of this \texttt{--backup} option is to save a persistent backup in case the actual content of the update causes problems, e.g., introduces a TeX incompatibility.

The Section B.6.26 [restore], page 64, action explains how to restore from a backup.
Appendix B: tlmgr

--no-depends
If you call for updating a package normally all depending packages will also be checked for updates and updated if necessary. This switch suppresses this behavior.

--no-depends-at-all
See above under Section B.6.13 [install], page 56, (and beware).

--force
Force update of normal packages, without updating tlmgr itself (unless the --self option is also given). Not recommended.
Also, update --list is still performed regardless of this option.

If the package on the server is older than the package already installed (e.g., if the selected mirror is out of date), tlmgr does not downgrade. Also, packages for uninstalled platforms are not installed.

tlmgr saves one copy of the main texlive.tlpdb file used for an update with a suffix representing the repository url, as in tlpkg/texlive.tlpdb.main.long-hash-string. Thus, even when many mirrors are used, only one main tlpdb backup is kept. For non-main repositories, which do not generally have (m)any mirrors, no pruning of backups is done.

This action does not automatically add or remove new symlinks in system directories; you need to run tlmgr Section B.6.18 [path], page 60, yourself if you are using this feature and want new symlinks added.

B.7 CONFIGURATION FILE FOR TLMGR

tlmgr reads two configuration files: one is system-wide, in TEXMFSYSCONFIG/tlmgr/config, and the other is user-specific, in TEXMFCONFIG/tlmgr/config. The user-specific one is the default for the conf tlmgr action. (Run kpsewhich -var-value=TEXMFSYSCONFIG or ...TEXMFCONFIG ... to see the actual directory names.)

A few defaults corresponding to command-line options can be set in these configuration files. In addition, the system-wide file can contain a directive to restrict the allowed actions.

In these config files, empty lines and lines starting with # are ignored. All other lines must look like:

key = value

where the spaces are optional but the = is required.

The allowed keys are:
auto-remove = 0 or 1 (default 1), same as command-line option.
gui-expertmode = 0 or 1 (default 1). This switches between the full GUI and a simplified GUI with only the most common settings.
gui-lang = llcode, with a language code value as with the command-line option.
no-checksums = 0 or 1 (default 0, see below).
persistent-downloads = 0 or 1 (default 1), same as command-line option.
require-verification = 0 or 1 (default 0), same as command-line option.
tkfontscale = floating-point number (default 1.0); scaling factor for fonts in the Tk-based frontends.
update-exclude = comma-separated list of packages (no spaces allowed). Same as the command line option --exclude for the update action.
verify-downloads = 0 or 1 (default 1), same as command-line option.

The system-wide config file can contain one additional key:

allowed-actions = action1[,action2,...] The value is a comma-separated list (no spaces) of \texttt{tlmgr} actions which are allowed to be executed when \texttt{tlmgr} is invoked in system mode (that is, without --usermode). This allows distributors to include \texttt{tlmgr} in their packaging, but allow only a restricted set of actions that do not interfere with their distro package manager. For native TeX Live installations, it doesn't make sense to set this.

Finally, the no-checksums key needs more explanation. By default, package checksums computed and stored on the server (in the TLPDB) are compared to checksums computed locally after downloading. no-checksums disables this process. The checksum algorithm is SHA-512. Your system must have one of (looked for in this order) the Perl Digest::SHA module, the openssl program (https://openssl.org), the sha512sum program (from GNU Coreutils, https://www.gnu.org/software/coreutils), or finally the shasum program (just to support old Macs). If none of these are available, a warning is issued and \texttt{tlmgr} proceeds without checking checksums. no-checksums avoids the warning. (Incidentally, other SHA implementations, such as the pure Perl and pure Lua modules, are much too slow to be usable in our context.)

### B.8 CRYPTOGRAPHIC VERIFICATION

\texttt{tlmgr} and \texttt{install-tl} perform cryptographic verification if possible. If verification is performed and successful, the programs report (verified) after loading the TLPDB; otherwise, they report (not verified). But either way, by default the installation and/or updates proceed normally.

If a program named \texttt{gpg} is available (that is, found in PATH), by default cryptographic signatures will be checked: we require the main repository be signed, but not any additional repositories. If \texttt{gpg} is not available, by default signatures are not checked and no verification is carried out, but \texttt{tlmgr} still proceeds normally.

The behavior of the verification can be controlled by the command line and config file option verify-repo which takes one of the following values: none, main, or all. With none, no verification whatsoever is attempted. With main (the default) verification is required only for the main repository, and only if \texttt{gpg} is available; though attempted for all, missing signatures of subsidiary repositories will not result in an error. Finally, in the case of all, \texttt{gpg} must be available and all repositories need to be signed.

In all cases, if a signature is checked and fails to verify, an error is raised.
Cryptographic verification requires checksum checking (described just above) to succeed, and a working GnuPG (gpg) program (see below for search method). Then, unless cryptographic verification has been disabled, a signature file (texlive.tlpdb.*.asc) of the checksum file is downloaded and the signature verified. The signature is created by the TeX Live Distribution GPG key 0x0D5E5D9106BAB6BC, which in turn is signed by Karl Berry’s key 0x0716748A30D155AD and Norbert Preining’s key 0x6CACA48860CDC13. All of these keys are obtainable from the standard key servers.

Additional trusted keys can be added using the key action.

**B.8.1 Configuration of GnuPG invocation**

The executable used for GnuPG is searched as follows: If the environment variable TL_GNUPG is set, it is tested and used; otherwise gpg is checked; finally gpg2 is checked.

Further adaptation of the gpg invocation can be made using the two environment variables TL_GNUPGHOME, which is passed to gpg as the value for --homedir, and TL_GNUPGARGS, which replaces the default options --no-secmem-warning --no-permission-warning.

**B.9 USER MODE**

**tlmgr** provides a restricted way, called “user mode”, to manage arbitrary texmf trees in the same way as the main installation. For example, this allows people without write permissions on the installation location to update/install packages into a tree of their own.

**tlmgr** is switched into user mode with the command line option --usermode. It does not switch automatically, nor is there any configuration file setting for it. Thus, this option has to be explicitly given every time user mode is to be activated.

This mode of **tlmgr** works on a user tree, by default the value of the TEXMFHOME variable. This can be overridden with the command line option --usertree. In the following when we speak of the user tree we mean either TEXMFHOME or the one given on the command line.

Not all actions are allowed in user mode; **tlmgr** will warn you and not carry out any problematic actions. Currently not supported (and probably will never be) is the platform action. The gui action is currently not supported, but may be in a future release.

Some **tlmgr** actions don’t need any write permissions and thus work the same in user mode and normal mode. Currently these are: check, help, list, print-platform, print-platform-info, search, show, version.

On the other hand, most of the actions dealing with package management do need write permissions, and thus behave differently in user mode, as described below: install, update, remove, option, paper, generate, backup, restore, uninstall, symlinks.

Before using **tlmgr** in user mode, you have to set up the user tree with the init-usertree action. This creates usertree/web2c and usertree/tlpkg/tlpobj, and a minimal usertree/tlpkg/texlive.tlpdb. At that point, you can tell **tlmgr** to do the (supported) actions by adding the --usermode command line option.

In user mode the file usertree/tlpkg/texlive.tlpdb contains only the packages that have been installed into the user tree using **tlmgr**, plus additional options from the “virtual” package 00texlive.installation (similar to the main installation’s texlive.tlpdb).

All actions on packages in user mode can only be carried out on packages that are known as relocatable. This excludes all packages containing executables and a few other core
packages. Of the 2500 or so packages currently in TeX Live the vast majority are relocatable and can be installed into a user tree.

Description of changes of actions in user mode:

**B.9.1 User mode install**

In user mode, the `install` action checks that the package and all dependencies are all either relocated or already installed in the system installation. If this is the case, it unpacks all containers to be installed into the user tree (to repeat, that’s either `TEXMFHOME` or the value of `--usertree`) and add the respective packages to the user tree’s `texlive.tlpdb` (creating it if need be).

Currently installing a collection in user mode installs all dependent packages, but in contrast to normal mode, does *not* install dependent collections. For example, in normal mode `tlmgr install collection-context` would install `collection-basic` and other collections, while in user mode, *only* the packages mentioned in `collection-context` are installed.

If a package shipping map files is installed in user mode, a backup of the user’s `updmap.cfg` in `USERTREE/web2c/` is made, and then this file regenerated from the list of installed packages.

**B.9.2 User mode backup, restore, remove, update**

In user mode, these actions check that all packages to be acted on are installed in the user tree before proceeding; otherwise, they behave just as in normal mode.

**B.9.3 User mode generate, option, paper**

In user mode, these actions operate only on the user tree’s configuration files and/or `texlive.tlpdb`.

**B.9.4 User mode logs**

In user mode, `tlmgr.log` and `<tlmgr-commands.log>` are written in the `TEXMFVAR/web2c/` directory instead of `TEXMFSYSVAR/web2c/`.

**B.10 MULTIPLE REPOSITORIES**

The main TeX Live repository contains a vast array of packages. Nevertheless, additional local repositories can be useful to provide locally-installed resources, such as proprietary fonts and house styles. Also, alternative package repositories distribute packages that cannot or should not be included in TeX Live, for whatever reason.

The simplest and most reliable method is to temporarily set the installation source to any repository (with the `-repository` or `option repository` command line options), and perform your operations.

When you are using multiple repositories over a sustained length of time, however, explicitly switching between them becomes inconvenient. Thus, it’s possible to tell `tlmgr` about additional repositories you want to use. The basic command is `tlmgr repository add`. The rest of this section explains further.

When using multiple repositories, one of them has to be set as the main repository, which distributes most of the installed packages. When you switch from a single repository
installation to a multiple repository installation, the previous sole repository will be set as the main repository.

By default, even if multiple repositories are configured, packages are still only installed from the main repository. Thus, simply adding a second repository does not actually enable installation of anything from there. You also have to specify which packages should be taken from the new repository, by specifying so-called “pinning” rules, described next.

B.10.1 Pinning

When a package foo is pinned to a repository, a package foo in any other repository, even if it has a higher revision number, will not be considered an installable candidate.

As mentioned above, by default everything is pinned to the main repository. Let’s now go through an example of setting up a second repository and enabling updates of a package from it.

First, check that we have support for multiple repositories, and have only one enabled (as is the case by default):

```
$ tlmgr repository list
List of repositories (with tags if set):
/var/www/norbert/tlnet
Ok. Let’s add the tlcontrib repository (this is a real repository hosted at http://contrib.texlive.info) with the tag tlcontrib:
```

```
$ tlmgr repository add http://contrib.texlive.info/current tlcontrib
Check the repository list again:
```

```
$ tlmgr repository list
List of repositories (with tags if set):
    http://contrib.texlive.info/current (tlcontrib)
    /var/www/norbert/tlnet (main)
Now we specify a pinning entry to get the package classico from tlcontrib:
```

```
$ tlmgr pinning add tlcontrib classico
Check that we can find classico:
```

```
$ tlmgr show classico
package: classico
...  
    shortdesc: URW Classico fonts
...  
    - install classico:
$ tlmgr install classico
    tlmgr: package repositories:
...  
[1/1, ???:??/??:??] install: classico @tlcontrib [737k]
In the output here you can see that the classico package has been installed from the tlcontrib repository (@tlcontrib).

Finally, tlmgr pinning also supports removing certain or all packages from a given repository:
```

```
$ tlmgr pinning remove tlcontrib classico # remove just classico
```
$ tlmgr pinning remove tlcontrib --all  # take nothing from tlcontrib
A summary of \texttt{tlmgr pinning} actions is given above.

\section*{B.11 GUI FOR TLMGR}

The graphical user interface for \texttt{tlmgr} requires Perl/Tk \url{https://search.cpan.org/search?query=perl%2Ftk}. For Unix-based systems Perl/Tk (as well as Perl of course) has to be installed outside of TL. \url{https://tug.org/texlive/distro.html#perltk} has a list of invocations for some distros. For Windows the necessary modules are no longer shipped within TeX Live, so you'll have to have an external Perl available that includes them.

We are talking here about the GUI built into tlmgr itself, not about the other tlmgr GUIs, which are: tlshell (Tcl/Tk-based), tlcockpit (Java-based) and, only on MacOS, TeX Live Utility. These are invoked as separate programs.

The GUI mode of tlmgr is started with the invocation \texttt{tlmgr gui}; assuming Tk is loadable, the graphical user interface will be shown. The main window contains a menu bar, the main display, and a status area where messages normally shown on the console are displayed.

Within the main display there are three main parts: the Display configuration area, the list of packages, and the action buttons.

Also, at the top right the currently loaded repository is shown; this also acts as a button and when clicked will try to load the default repository. To load a different repository, see the \texttt{tlmgr} menu item.

Finally, the status area at the bottom of the window gives additional information about what is going on.

\subsection*{B.11.1 Main display}

\subsubsection*{B.11.1.1 Display configuration area}

The first part of the main display allows you to specify (filter) which packages are shown. By default, all are shown. Changes here are reflected right away.

\begin{itemize}
  \item \textbf{Status}\n  \begin{itemize}
    \item Select whether to show all packages (the default), only those installed, only those \textit{not} installed, or only those with update available.
  \end{itemize}
  \item \textbf{Category}\n  \begin{itemize}
    \item Select which categories are shown: packages, collections, and/or schemes. These are briefly explained in the Section B.3 [DESCRIPTION], page 46, section above.
  \end{itemize}
  \item \textbf{Match}\n  \begin{itemize}
    \item Select packages matching for a specific pattern. By default, this searches both descriptions and filenames. You can also select a subset for searching.
  \end{itemize}
  \item \textbf{Selection}\n  \begin{itemize}
    \item Select packages to those selected, those not selected, or all. Here, “selected” means that the checkbox in the beginning of the line of a package is ticked.
  \end{itemize}
\end{itemize}
Display configuration buttons

To the right there are three buttons: select all packages, select none (a.k.a. deselect all), and reset all these filters to the defaults, i.e., show all available.

**B.11.1.2 Package list area**

The second area of the main display lists all installed packages. If a repository is loaded, those that are available but not installed are also listed.

Double clicking on a package line pops up an informational window with further details: the long description, included files, etc.

Each line of the package list consists of the following items:

- a checkbox
  
  Used to select particular packages; some of the action buttons (see below) work only on the selected packages.

- package name
  
  The name (identifier) of the package as given in the database.

- local revision (and version)
  
  If the package is installed the TeX Live revision number for the installed package will be shown. If there is a catalogue version given in the database for this package, it will be shown in parentheses. However, the catalogue version, unlike the TL revision, is not guaranteed to reflect what is actually installed.

- remote revision (and version)
  
  If a repository has been loaded the revision of the package in the repository (if present) is shown. As with the local column, if a catalogue version is provided it will be displayed. And also as with the local column, the catalogue version may be stale.

- short description
  
  The short description of the package.

**B.11.1.3 Main display action buttons**

Below the list of packages are several buttons:

- **Update all installed**
  
  This calls `tlmgr update --all`, i.e., tries to update all available packages. Below this button is a toggle to allow reinstallation of previously removed packages as part of this action.

  The other four buttons only work on the selected packages, i.e., those where the checkbox at the beginning of the package line is ticked.

- **Update**
  
  Update only the selected packages.

- **Install**
  
  Install the selected packages; acts like `tlmgr install`, i.e., also installs dependencies. Thus, installing a collection installs all its constituent packages.
Appendix B: tlmgr

Remove

Removes the selected packages; acts like `tlmgr remove`, i.e., it will also remove dependencies of collections (but not dependencies of normal packages).

Backup

Makes a backup of the selected packages; acts like `tlmgr backup`. This action needs the option `backupdir` set (see Options - General).

B.11.2 Menu bar

The following entries can be found in the menu bar:

**tlmgr menu**

The items here load various repositories: the default as specified in the TeX Live database, the default network repository, the repository specified on the command line (if any), and an arbitrarily manually-entered one. Also has the so-necessary `quit` operation.

**Options menu**

Provides access to several groups of options: Paper (configuration of default paper sizes), Platforms (only on Unix, configuration of the supported/installed platforms), GUI Language (select language used in the GUI interface), and General (everything else).

Several toggles are also here. The first is Expert options, which is set by default. If you turn this off, the next time you start the GUI a simplified screen will be shown that display only the most important functionality. This setting is saved in the configuration file of `tlmgr`; see Section B.7 [CONFIGURATION FILE FOR TLMGR], page 69, for details.

The other toggles are all off by default: for debugging output, to disable the automatic installation of new packages, and to disable the automatic removal of packages deleted from the server. Playing with the choices of what is or isn’t installed may lead to an inconsistent TeX Live installation; e.g., when a package is renamed.

**Actions menu**

Provides access to several actions: update the filename database (aka `ls-R`, `mktexlsr`, `texhash`), rebuild all formats (`fmtutil-sys --all`), update the font map database (`updmap-sys`), restore from a backup of a package, and use of symbolic links in system directories (not on Windows).

The final action is to remove the entire TeX Live installation (also not on Windows).

**Help menu**

Provides access to the TeX Live manual (also on the web at https://tug.org/texlive/doc.html) and the usual “About” box.

B.11.3 GUI options

Some generic Perl/Tk options can be specified with `tlmgr gui` to control the display:

```
-background color
```

Set background color.
Appendix B: tlmgr

-\texttt{font} "\texttt{fontname fontsize}"

Set font, e.g., \texttt{tlmgr gui -font} "\texttt{helvetica 18}". The argument to \texttt{font} must
be quoted, i.e., passed as a single string.

-\texttt{foreground color}

Set foreground color.

-\texttt{geometry geomspec}

Set the X geometry, e.g., \texttt{tlmgr gui -geometry 1024x512-0+0} creates the window
of (approximately) the given size in the upper-right corner of the display.

-\texttt{xrm xresource}

Pass the arbitrary X resource string \texttt{xresource}.

A few other obscure options are recognized but not mentioned here. See the Perl/Tk
documentation (https://search.cpan.org/perldoc?Tk) for the complete list, and any X
documentation for general information.

\subsection{B.12 MACHINE-READABLE OUTPUT}

With the \texttt{--machine-readable} option, \texttt{tlmgr} writes to stdout in the fixed line-oriented
format described here, and the usual informational messages for human consumption are
written to stderr (normally they are written to stdout). The idea is that a program can get
all the information it needs by reading stdout.

Currently this option only applies to the Section B.6.31 [update], page 66, Section B.6.13
[install], page 56, and Section B.6.16 [option], page 58, actions.

\subsubsection{B.12.1 Machine-readable update and install output}

The output format is as follows:

\begin{verbatim}
fieldname "\t" value ...
"end-of-header"
pkgname status localrev serverrev size runtime esttot ...
"end-of-updates"
other output from post actions, not in machine readable form
\end{verbatim}

The header section currently has two fields: \texttt{location-url} (the repository source from
which updates are being drawn), and \texttt{total-bytes} (the total number of bytes to be down-
loaded).

The \texttt{localrev} and \texttt{serverrev} fields for each package are the revision numbers in the local
installation and server repository, respectively. The \texttt{size} field is the number of bytes to
be downloaded, i.e., the size of the compressed tar file for a network installation, not the
unpacked size. The runtime and \texttt{esttot} fields are only present for updated and auto-install
packages, and contain the currently passed time since start of installation(updates and the
estimated total time.

Line endings may be either LF or CRLF depending on the current platform.
location-url location
The location may be a url (including file:///foo/bar/...), or a directory name (/foo/bar). It is the package repository from which the new package information was drawn.

total-bytes count
The count is simply a decimal number, the sum of the sizes of all the packages that need updating or installing (which are listed subsequently).

Then comes a line with only the literal string end-of-header.
Each following line until a line with literal string end-of-updates reports on one package. The fields on each line are separated by a tab. Here are the fields.

pkgname
The TeX Live package identifier, with a possible platform suffix for executables. For instance, pdftex and pdftex.i386-linux are given as two separate packages, one on each line.

status
The status of the package update. One character, as follows:

d
The package was removed on the server.

f
The package was removed in the local installation, even though a collection depended on it. (E.g., the user ran tlmgr remove --force.)

u
Normal update is needed.

r
Reversed non-update: the locally-installed version is newer than the version on the server.

a
Automatically-determined need for installation, the package is new on the server and is (most probably) part of an installed collection.

i
Package will be installed and isn’t present in the local installation (action install).

I
Package is already present but will be reinstalled (action install).

localrev
The revision number of the installed package, or - if it is not present locally.

serverrev
The revision number of the package on the server, or - if it is not present on the server.
Appendix B: \texttt{tlmgr}

\textbf{size}

The size in bytes of the package on the server. The sum of all the package sizes is given in the \texttt{total-bytes} header field mentioned above.

\textbf{runtime}

The run time since start of installations or updates.

\textbf{esttot}

The estimated total time.

\textbf{B.12.2 Machine-readable option output}

The output format is as follows:

\begin{verbatim}
key "\t" value
\end{verbatim}

If a value is not saved in the database the string \texttt{(not set)} is shown.

If you are developing a program that uses this output, and find that changes would be helpful, do not hesitate to write the mailing list.

\textbf{B.13 ENVIRONMENT VARIABLES}

\texttt{tlmgr} uses many of the standard TeX environment variables, as reported by, e.g., \texttt{tlmgr conf} (Section B.6.7 [conf], page 52).

In addition, for ease in scripting and debugging, \texttt{tlmgr} looks for the following environment variables. These are not of interest for normal user installations.

\texttt{TEXLIVE\_COMPRESSOR}

This variable allows selecting a different compressor program for backups and intermediate rollback containers. The order of selection is:

1. If the environment variable \texttt{TEXLIVE\_COMPRESSOR} is defined, use it; abort if it doesn’t work. Possible values: \texttt{lz4}, \texttt{gzip}, \texttt{xz}. The necessary options are added internally.

2. If \texttt{lz4} is available (either from the system or TL) and working, use that.

3. If \texttt{gzip} is available (from the system) and working, use that.

4. If \texttt{xz} is available (either from the system or TL) and working, use that.

\texttt{lxz} and \texttt{gzip} are faster in creating \texttt{tlmgr}’s local backups, hence they are preferred. The unconditional use of \texttt{xz} for the \texttt{tlnet} containers is unaffected, to minimize download sizes.

\texttt{TEXLIVE\_DOWNLOADER}

\texttt{TL\_DOWNLOAD\_PROGRAM}

\texttt{TL\_DOWNLOAD\_ARGS}

These options allow selecting different download programs then the ones automatically selected by the installer. The order of selection is:

1. If the environment variable \texttt{TEXLIVE\_DOWNLOADER} is defined, use it; abort if the specified program doesn’t work. Possible values: \texttt{lwp}, \texttt{curl}, \texttt{wget}. The necessary options are added internally.
2. If the environment variable TL\_DOWNLOAD\_PROGRAM is defined (can be any value), use it together with TL\_DOWNLOAD\_ARGS; abort if it doesn’t work.
3. If LWP is available and working, use that (by far the most efficient method, as it supports persistent downloads).
4. If curl is available (from the system) and working, use that.
5. If wget is available (either from the system or TL) and working, use that.

TL provides \texttt{wget} binaries for platforms where necessary, so some download method should always be available.

\texttt{TEXLIVE\_PREFER\_OWN}

By default, compression and download programs provided by the system, i.e., found along \texttt{PATH} are preferred over those shipped with TeX Live.

This can create problems with systems that are too old, and so can be overridden by setting the environment variable \texttt{TEXLIVE\_PREFER\_OWN} to 1. In this case, executables shipped with TL will be preferred.

Extra compression/download programs not provided by TL, such as gzip, lwp, and curl, are still checked for on the system and used if available, per the above. \texttt{TEXLIVE\_PREFER\_OWN} only applies when the program being checked for is shipped with TL, namely the lz4 and xz compressors and wget downloader.

Exception: on Windows, the \texttt{tar.exe} shipped with TL is always used, regardless of any setting.

\textbf{B.14 AUTHORS AND COPYRIGHT}

This script and its documentation were written for the TeX Live distribution (\url{https://tug.org/texlive}) and both are licensed under the GNU General Public License Version 2 or later.

$Id: tlmgr.pl 70001 2024-02-19 23:17:07Z karl $

\textbf{B.15 POD ERRORS}

Hey! The above document had some coding errors, which are explained below:

Around line 8454:

Unterminated C<...> sequence
Index

$ §

$ target in normal make rules 3

bindir configure option 25, 26
build=host 9
disable-all-packages 6
disable-all-pkgs 27
disable-bibtex8 28
disable-bibtexu 28
disable-dump-share 27
disable-dvipdfmx 28
disable-etex-syntex 28
disable-ipc 27
disable-largefile 25
disable-linked-scripts 29
disable-mf-noxin 27
disable-missing 26
disable-native-texlive-build 25
disable-prog 27
disable-syntex 28
disable-tex 27
disable-web-progs 27
disable-xdvi-progs 28
enable-*win for Metafont window support 28
enable-auto-core 27
enable-compilation-warnings=level 26
enable-cxx-runtime-translate 16
enable-etex 27
enable-libtool-hack 28
enable-maintainer-mode 14, 26
enable-missing to ignore dependencies 7
enable-mktextfm-default 19
enable-multiplatform 26
enable-prog 27
enable-shared 26
enable-silent-rules 26
enable-tex-syntex 28
enable-texlive-build 25
enable-x11-scrolling 29
enable-xindy-docs 29
enable-xindy-rules 29
host=host 9
libdir configure option 26
no-print-directory GNU make option 8
prefix configure option 25
with-binder-add=strict 27
with-cisp-runtime=filename 29
with-editor=cmd 27
with-fontconfig-includes=dir 27
with-fontconfig-libdir=dir 27
with-gs=filename 29
with-lib-includes=dir, -libdir 29

--with-libgs-includes, -libdir 28
--with-system-kpathsea 19
--with-system-lib 23, 29
--with-system-libgs 28
--with-xdvi-x-toolkit 20
--with-xdvi-x-toolkit=kit 29
without-libs 28
without-ln-s 26
without-x 26
-C configure option 5
-j make option 5

A

ac/withenable_ac 22
adapting TeX Live for distros 13
adding a new engine 23
adding a new generic library 23
adding a new program 21
adding a new TeX-specific library 24
adding to TeX Live 21
am/ top-level directory 14
ANSI C 31
ApplicationServices Mac framework, required by 3
asymptote 12, 21
Autoconf 2
autoconf macros 15
Automake 2
autoreconf, for new program 22

B

biber 12
bibtex-x 28
bibtex8 28
bibtexu 28
BSD distro 12
build directory, required 5
build iteration 5
build on demand 6
build one engine 7
build one package 6
Build script 5
build system, design of 2
build-aux/ top-level directory 15
BUILDCC, BUILDCCFLAGS 9
building 5
building a distribution 5
building in parallel 5
C
C++11, removing dependent sources .................................. 7
C++11, required ......................................................... 3
C, ANSI, required ..................................................... 31
C99, avoided ............................................................ 31
cache file, for configure ............................................... 5
cache for configure .................................................... 5
callexc.c ................................................................. 18
CC .......................................................... 30
CC=c-compiler ......................................................... 7
CC_BUILD ............................................................ 9
Changelog .............................................................. 22
chktex ................................................................. 31
cilisp ................................................................. 30
cilisp, required by xindy ............................................ 3
CLISP ................................................................. 30
Cocoa Mac framework, required by \texttt{xetex} .............. 3
coding conventions .................................................... 31
compilers, C and C++11 ............................................... 3
configure \texttt{.g}uess, \texttt{.g}uess subsidized .................. 15
configure options ..................................................... 25
configure options, for \texttt{bibtex-x} ......................... 28
configure options, for \texttt{dvipdfm-x} ...................... 28
configure options, for \texttt{dvisvgm} ......................... 28
configure options, for \texttt{kpathsea} ....................... 29
configure options, for \texttt{texk/texlive} ................. 29
configure options, for \texttt{web2c} ............................ 27
configure options, for \texttt{xdvik} .......................... 29
configure options, for \texttt{xindy} .......................... 29
configure options, global .......................................... 25
configure options, library-specific .......................... 29
configure options, program-specific .......................... 27
configure problems, work around by removing ............ 7
configure variables .................................................. 30
configure.ac .......................................................... 22
configuring, for cross compilation .......................... 9
c\texttt{onst} .......................................................... 32
continuous integration .............................................. 33
c\texttt{onventions, coding} ........................................ 31
CPPFLAGS ............................................................ 30
cross compilation .................................................... 8
cross compilation configuring .................................. 9
cross compilation problems ..................................... 9
cross compilation, with host binary ......................... 20
c\texttt{tangle} .......................................................... 10
CXX ................................................................. 30
CXX=c++-compiler .................................................. 7
distribution tarball, making .................................... 5
distro, building for .................................................. 12
d\texttt{v}ipdf\texttt{m-x} ..................................................... 28
d\texttt{v}ipdf\texttt{m} ....................................................... 28
d\texttt{visvgm} .......................................................... 28
d\texttt{visvgm}, requires C++11 ................................... 3
E
engine, adding new .................................................. 23
engine, building one ............................................... 7
environment variables, for \texttt{configure} ................ 25
exec\texttt{.p}refix ....................................................... 26
ext\texttt{en}ting \texttt{T}ex \texttt{L}ive .................................... 21
ex\texttt{tern} functions ............................................... 31
F
flags, macros for library and header ......................... 17
font\texttt{c}onfig library, required by \texttt{xetex} .......... 3
f\texttt{reetype} cross compiling .................................. 9
f\texttt{reetype} library ................................................ 19
Free\texttt{Type}, requires \texttt{gmake} .......................... 3
f\texttt{reetype-cong} .................................................. 19, 30
FT2\texttt{.c}ONFIG ....................................................... 30
G
g\texttt{cc}, default compilers ....................................... 7
general setup macros .............................................. 15
generic library module, adding ............................... 23
Ghostscript location for Xdvik ................................ 29
g\texttt{it-svn} ............................................................. 33
global configure options ........................................ 25
\texttt{gmake}, required ............................................... 3
GNU \texttt{make}, required .......................................... 3
GNU tools, needed for building ............................... 14
GNU/Linux distro ................................................... 12
Gnum\texttt{lib}, used for common files ......................... 15
H
HarfBuzz, requires C++11 ........................................ 3
<table>
<thead>
<tr>
<th>X</th>
<th>xindy</th>
<th>12, 20, 29</th>
</tr>
</thead>
<tbody>
<tr>
<td>X toolkit</td>
<td>xindy cross compiling requires clisp</td>
<td>10</td>
</tr>
<tr>
<td>X11 development, required by X clients</td>
<td>XInput</td>
<td>29</td>
</tr>
<tr>
<td>X11 headers, and const</td>
<td>Xlib</td>
<td>27</td>
</tr>
<tr>
<td>xasy</td>
<td>xz</td>
<td>12</td>
</tr>
<tr>
<td>xaw</td>
<td>zlib library</td>
<td>19</td>
</tr>
<tr>
<td>xdvik</td>
<td>zlib library</td>
<td>19</td>
</tr>
<tr>
<td>xdvipdfmx</td>
<td>xindy</td>
<td>12, 20, 29</td>
</tr>
</tbody>
</table>