Building TeX Live (2017)

Peter Breitenlohner
Karl Berry
http://tug.org/texlive

This file documents the TeX Live build system and more.

Short Contents

Table of Contents

1 Introduction

This manual (dated April 2017) corresponds to the TeX Live 2017 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 `source/README` file in the TL source tree provides the 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 `http://tug.org/texlive/svn`.

This manual does not duplicate the (primarily user-level) information found in other TL documentation resources, such as:

- The TeX Live web pages: `http://tug.org/texlive`.
- The web page with an overview of how to build the binaries which are distributed with TeX Live: `http://tug.org/texlive/build.html`.
- The TeX Live user manual: `http://tug.org/texlive/doc.html`, or run `texdoc texlive`.
- Other TeX-related Texinfo manuals (see *Web2c*, *Kpathsea*, etc.): `http://tug.org/texinfohtml/`, or check the 'TeX' category in the GNU Info system.
- Package documentation: `http://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, just because it is convenient to do so. The same text is available online (linked from `http://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, consistently using Autoconf, Automake, and Libtool. Thus

    configure && make && make check && make install

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

The main components of the TL build system are:

`libs/`*lib*   Generic libraries.

`texk/`*lib*   TeX-specific libraries in subdirectories, notably *lib*=`kpathsea`. (The other one is `texk/ptexenc`.)

`texk/`*prog*

>   TeX-specific programs (that use Kpathsea).

`utils/`*prog*

>   Other programs (that don't use Kpathsea).

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, `configure` options to be seen at the top level, and more. An explicit list of all available modules is kept in only one central place, namely `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. (Not to say that either task is trivial.)

3 Prerequisites

Overall, building the TeX Live programs, when using all libraries from the TL source tree, requires only C and C++ compilers and GNU `make`. (If `make` from your `PATH` is not GNU make, you can set `MAKE` in the environment to whatever is necessary.)

GNU `make` is required only because of some third-party libraries, notably FreeType; all the TL-maintained directories (and Automake/Autoconf output in general) should work with any reasonable `make`.

However, a few programs in the tree have additional requirements:

dvisvgm   requires a C++11 compiler, such as gcc 4.8.1 (or later) or clang 3.3 (or later).

web2c   requires `perl` for some tests run by `make check`.

xdvik
xpdfopen   require X11 headers and libraries, often in "development" packages that are not installed by default.

xetex   requires `fontconfig` (again both headers and library), or, for MacOSX only, the `ApplicationServices` and `Cocoa` frameworks.

xindy   requires GNU `clisp` and in addition `perl`, `latex`, and `pdflatex` to build the rules and/or documentation.

Lacking the required tools, building these programs must avoided, e.g.,
`configure --without-x --disable-xetex --disable-xindy`

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

- Modification of any `.y` or `.l` source files requires `bison` or `flex` to updatete the corresponding C sources.
- Modification of the sources for `.info` files requires `makeinfo`.
- Modification of any part of the build system (M4 macros, `configure.ac`, `Makefile.am`, or their fragments) requires GNU M4, GNU Autoconf, GNU Automake, and GNU Libtool to update the generated files. See Section 6.1 [Build system tools], page 10.

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 10). Barring buggy commits, no infrastructure tools are needed to do a normal build.

As an example, on Debian systems the necessary build dependencies can be installed via:

```
apt-get install libfontconfig-dev libx11-dev libxmu-dev libxaw7-dev
```

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 configures and makes everything in a subdirectory of the main build tree (default `Work/`), installs everything in another subdirectory (default `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 itself 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` oneself 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 `libs`, `utils`, and `texk`. Running `make` at the top-level first iterates over all TeX-specific libraries, and then runs `make` in `libs`, `utils`, and `texk` to iterate over all generic libraries, utility programs, and TeX-specific programs. These iterations consist of two steps:

1. For each library or program module not yet configured, run `configure`, adding the configure option `--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): `default` or `all` to (re-)build, `check` to run tests, `install`, etc.

Running the top-level `make` a second time iterates again over all the library and program modules, but finds (should find) nothing to be done unless some source files have been modified.

4.2 Build problems

If configuring or building a module fails, you should first find and fix the problem, then perhaps remove the subdirectory for that module from the build tree, and finally rerun the top level `make` (or `Build` with `--no-clean` as its first argument).

4.3 Build in parallel

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

Incidentally, a noticeable speed-up can also be (independently) gained by using a configure cache file, i.e., with the option `-C` (recommended).

4.4 Build distribution

Running `make dist` at the top level creates a tarball `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 7.

4.5 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 21). 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 and causes to first build any required libraries.

This "build-on-demand" procedure is used, e.g., in the upstream LuaTeX repository to build LuaTeX, essentially from a subset of the complete TeX Live tree. Similarly, when, e.g., building the original e-TeX has been disabled (as it is by default), one can run `make etex` (or `make etex.exe`) in `texk/web2c/` to build e-TeX (although there is no comparably simple way to install e-TeX).

If you want to work on a single program within the TL sources, this is the recommended way to do it. Here is an example from start to finish for working on `dvipdfm-x`.

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

# Get sources (http://tug.org/texlive/svn)
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

# Do the make:
make >&outm

# Test:
cd texk/dvipdfm-x
make check
```

Then you modify source files in `mydir/texk/dvipdfm-x` and rerun `make` in `mydir/Work/texk/dvipdfm-x` to rebuild.

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 up `configure`, and the `CFLAGS` and `CXXFLAGS` settings eliminate compiler optimization for debugging purposes.

Of course, one should 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`.

Finally, 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, see the `build-pdftex.sh` script in the `pdftex` development source (`http://pdftex.org`), which is indeed a cut-down TL source tree.

Even with `--disable-all-pkgs`, dependencies will be checked. For instance, if a non-MacOSX system does not have `fontconfig`, XeTeX cannot be built (see Chapter 3 [Prerequisites], page 3) and `configure` will terminate. To proceed without such dependencies, specify `--enable-missing` also. (Arguably this should happen automatically.)

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`.

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 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 `http://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 7) 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 explictly as options such as `--prefix=`*prefix* or `--bindir=`*bindir*; otherwise, they are given by their usual Autoconf defaults:

prefix	`/usr/local`
exec_prefix	*prefix*
bindir	*exec_prefix*/`bin`
libdir	*exec_prefix*/`lib`
includedir	*prefix*/`include`
datarootdir	*prefix*/`share`
mandir	*datarootdir*/`man`
infodir	*datarootdir*/`info`

except possibly modified as follows:

- If the option `--enable-multiplatform` 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-platform-name*. 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 (e.g., *bindir*`/ps2eps.exe`) serves the same purpose. (The source for the wrapper is in `texk/texlive/w32_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 `http://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`). However, 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, but we digress). 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:

`--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*
> Use system versions for as many libraries *lib* as possible. `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 7), 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: `http://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 requires very little. 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) 2.69
automake (GNU automake) 1.15
bison (GNU Bison) 3.0.4
flex 2.6.0
ltmain.sh (GNU libtool) 2.4.6
m4 (GNU M4) 1.4.18
makeinfo (GNU texinfo) 6.1

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`.

The files in the Subversion repository (see `http://tug.org/texlive/svn`) are all up to date, but unfortunately this may not be reflected by their timestamps. (For starters, be sure to set `use-commit-times=yes` in `~/.subversion/config` or the equivalent.)

To avoid unnecessary runs of `bison`, `flex`, or `makeinfo` it may be necessary to `touch` 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. Perhaps `make -t` will help.

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`, `depcomp`, etc. used by most packages. These are from the GNU Gnulib sources (`http://www.gnu.org/software/gnulib`), which in turn synchronizes with the appropriate ultimate upstream repository. There are, however, independent copies in, e.g., `libs/freetype2/freetype-*/builds/unix/`, and a few other places. The `reautoconf` script does not touch those, but a TL cron job keeps them in sync (nightly).

The directory `extra/` contains things which are not part of the TL build, but are present just for (someone's) convenience, e.g., is `epstopdf` development source is here.

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 file.

6.3 Autoconf macros

Here we describe some of the Autoconf macros used in several modules–not a complete list, by any means. These general macros are supplemented by module-specific macros in directories such as `texk/dvipng/m4/`; some of those are described in Section 6.4 [Library modules], page 14, and see Section 6.5 [Program modules], page 16.

6.3.1 General setup macros

The TL sources use two general setup macros:

`KPSE_BASIC (`*name*`, [`*more-options*`])`   [Macro]
>   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 (`*name*`, [`*more-options*`])`   [Macro]
>   Like `KPSE_BASIC` but add:
>   `LT_PREREQ([2.2.6])`
>   `LT_INIT([win32-dll])`
>   `AC_SYS_LARGEFILE`
>   `AC_FUNC_FSEEKO`
>   and check 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   [Macro]
> Set `LATEX` to the name of 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_PDFLATEX   [Macro]
> Check for `pdflatex` in `PATH` and set `PDFLATEX`.

KPSE_CHECK_PERL   [Macro]
> Check for `perl` or `perl5` in `PATH` and set `PERL`.

KPSE_PROG_LEX   [Macro]
> Call `AC_PROG_LEX` and add the flag `-l` for `flex`.

6.3.3 Macros for compilers

Macros for compiler-related checks:

KPSE_COMPILER_WARNINGS   [Macro]
> 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 `--enable-compiler-warnings`). Call `AC_SUBST` for them. At the moment this only works for GNU compilers, but could be extended to others if necessary.
>
> This macro caches its results in the `kpse_cv_warning_cflags`, ... variables.

KPSE_COMPILER_VISIBILITY   [Macro]
> 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 the moment this only tests for the compiler option `-fvisibility=hidden`, but that could be extended with more checks if necessary.
>
> This macro caches its results in the `kpse_cv_visibility_cflags` or `kpse_cv_visibility_cxxflags` variable.

KPSE_CXX_HACK   [Macro]
> Provide the configure option `--enable-cxx-runtime-hack`. If enabled and when using `g++`, try to statically link with `libstdc++`, somewhat improving portability of the resulting binary.
>
> This macro caches its result in the `kpse_cv_cxx_hack` variable.

6.3.4 Macros for libraries

One macro for a library check:

KPSE_LARGEFILE (*variable*, *[extra-define]*)   [Macro]
> Call `AC_SYS_LARGEFILE` and `AC_FUNC_FSEEKO` and append suitable `-D` flags (optionally including `-D`*extra-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 dependency, and `LIBPNG_RULE` defining `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:

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

```
KPSE_ADD_FLAGS([zlib])
... # tests for zlib features, if any
KPSE_ADD_FLAGS([libpng])
... # tests for libpng features
KPSE_RESTORE_FLAGS # restore CPPFLAGS and LIBS
```

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 `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/w32_wrapper/runscript.exe`) exists. This wrapper is used on Windows instead of symlinks for the "linked scripts" (see Section 5.2 [Linked scripts], page 7).

KPSE_WIN32_CALL											[Macro]

>   Call `KPSE_COND_WIN32`, check if the file `texk/texlive/w32_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`

This generic library uses the source tree in, e.g., 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])
```

specifying the module name, and indicating 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);])
```

thus providing 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);
  return 0; }
```

which Autoconf uses to verify the usability of a system version with C code. The analogous macro `KPSE_TRY_LIBXX` would check using C++ code. These fragments are included by `configure.ac` at the top level.

For this library, among many other modules, a proxy build system for TL is used (`configure.ac`, `Makefile.am`, and `include/Makefile.am`), ignoring the distributed one. 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` defining `make` rules to rebuild the library.

`m4/kpse-libpng-flags.m4` also supplies the configure option `--with-system-libpng` and 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 with an almost trivial `configure.ac` and with a `Makefile.am` that invokes `configure` and `make` for the distributed source, followed by `make install` with the build tree as destination. 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*). Despite being a core part of the TeX system, it is not a terribly special case in the infrastructure. The 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 perhaps even useful for distro builds (see Section 5.3 [Distro builds], page 8), 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 `ac/withenable.ac` and `ac/kpathsea.ac` there is a third fragment `ac/mktex.ac` included by both `ac/withenable.ac` and `configure.ac` that supplies configure options such as `--enable-mktextfm-default`, which determine the compile time default of whether or not to run `mktextfm` to generate a missing `.tfm` 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 14), here we discuss the details
for a few of the programs in TL.

6.5.1 The `t1utils` package in `utils/t1utils`

Once again 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/*`, a proxy `configure.ac`, and a wrapper `Makefile.am` that descends into
`xindy-src`. The `xindy` build requires a `make` that supports a `VPATH` build, can han-
dle all targets, and do not refer to `${top_srcdir}` or `${top_builddir}`. The fragment
`ac/withenable.ac` contains

```
KPSE_ENABLE_PROG([xindy], , [disable native])
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 en-
abled by the user with `configure --enable-xindy` (the need for `clisp` is too painful to
require by default), and `native` disallows cross compilation. 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 top level directory
and the subdirectory `gui`. The fragment `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 the dependency 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
and in a cross compilation they 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.

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, as modified for TL, in `foo/foo-src`, and the changes should be documented in `foo/TLpatches/*`; changes should also be submitted upstream whenever reasonable, of course. In addition, `foo/` will need the usual Automake build-related files (`configure.ac`, `Makefile.am`, etc. Please keep a `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_texk_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, apart from the program sources and build system (`configure.ac` and `Makefile.am`), 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. and 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`.

6.6.2 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 list `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 (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. Furthermore 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])`

6.6.3 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 list `kpse_texlibs_pkgs` 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 assigments 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 4). The defaults discussed below are those for the actual configure script; invoking configure via ./Build may yield different results.

Defaults for most options are set at the top level and propagated explicitly to all sub-directories. 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 would be used (see Section 5.3 [Distro builds], page 8). This may fail without GNU make, but will be tried regardless.

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) can use this option, e.g., to choose TL-specific 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 7.

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), needed on most 32-bit Unix systems for files with 2GB or more. Regardless of this, 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`. The size of PDF images included by `pdftex` or `luatex` must, however, be < 2GB when using `poppler` version 0.22 or before (even on 64-bit systems with LFS).

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 (Objective) C and 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 `WARNING_[OBJ]C[XX]FLAGS` but these flags are not used in all library and program modules. Using them should help to resolve portability problems.

  At present, these warning flags are only defined for the GNU compilers but flags for other compilers could be added when needed.

7.1.6 `--enable-cxx-runtime-hack`

If enabled (as it is for the native TL build) and when using `g++`, try to statically link with `libstdc++`, somewhat improving portability of the resulting binary. See Section 6.3.3 [Macros for compilers], page 12.

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 10), 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 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 another `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 even fail.

7.1.12 `--without-x`

Disable all programs using the X Window System.

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 the 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. The `configure` will generally take less than a minute with everything disabled. (It is a good idea to run `make check` after doing this, and after making any changes, to ensure that whatever tests have been written still pass.)

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 (`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-like engines, 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 core 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 other window support for Metafont (EPSF output, `mftalk`, old 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 again 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 with 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, or otherwise 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 7), 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`).

`--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 for a non-native one) 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 (if any) 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 whether or not to run *cmd*, one of:

`mkocp`   (Omega compiled translation process file)

`mkofm`   (Omega font metrics file)

`mktexfmt`   (format/base dump file)

`mktexmf`   (Metafont source)

`mktexpk`   (PK bitmap font)

`mktextex`   (TEX source)

`mktextfm`   (TFM file)

to generate the specified type of file dynamically. The default can be overridden by the user in any case.

7.3.2 Configure options for system `poppler`

Building LuaTEX (or LuaJITTEX) and XeTEX requires `poppler`, either from the TL tree or system headers and library. Building pdfTEX requires either `xpdf` from the TEX Live tree or system `poppler` headers and library.

`--with-system-poppler`
Use a system version (0.18 or newer) of `poppler` for LuaTEX (or LuaJITTEX) and XeTEX, and use `pkg-config` to obtain the required flags.

`--with-system-xpdf`
Use a system version (0.12 or better) of `poppler` (and `pkg-config`) for pdfTEX instead of `xpdf` from the TL tree. See Section 7.1.3 [`--disable-largefile`], page 19.

7.4 Variables for configure

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

`CC`
`CXX`
`CPPFLAGS`   And plenty more. As usual with Autoconf, these variables specify the name (or full path) of compilers, preprocessor flags, and similar. See Section "autoconf" in GNU `Autoconf`.

`CLISP`   Name (or full path) of the `clisp` executable, used to build `xindy`.

`FT2_CONFIG`
`ICU_CONFIG`
`PKG_CONFIG`
             These specify the name (or path) for the `freetype-config`, `icu-config`, and `pkg-config` commands used to determine the flags required for system versions of `libfreetype`, the ICU libraries, or many other libraries.

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

MAKE
SED   And more. Name (or path) of the `make`, `sed`, and similar programs; used at the top level and propagated to all subdirectories.

PERL
LATEX
PDFLATEX Name (or full path) for the `perl`, `latex`, and `pdflatex` commands used, e.g., to build the `xindy` documentation.

8 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 specify 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 MacOSX system one could use:

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

8.1 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:

```
ar                AR
freetype-config   FT2_CONFIG
g++               CXX
gcc               CC
icu-config        ICU_CONFIG
objdump           OBJDUMP
pkg-config        PKG_CONFIG
ranlib            RANLIB
strip             STRIP
```

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=...
```

8.2 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/xdvik/squeeze`. The module `libs/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 `libs/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 the WEB or CWEB programs requires sufficiently recent installed versions of `tangle`, `ctangle`, `otangle`, and `tie`.

Building `xindy` requires running the host system `clisp` binary, thus cross compilation is not possible.

9 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 most of the TeX Live tree, the exception being code that is maintained independently and whose maintainers don't want to accept patches.

9.1 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 the perhaps slightly less efficient alternative

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

in the file `Utility.h`.

Static functions

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

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.

9.2 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, and the same is true to some extent for `libfreetype` (but, thankfully, not for `zlib` nowadays).

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 headers or macros 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.

Appendix A install-tl

A.1 NAME

install-tl - TeX Live cross-platform installer

A.2 SYNOPSIS

install-tl [*option*]...

    install-tl.bat [*option*]...

A.3 DESCRIPTION

This installer creates a runnable TeX Live installation from various media, including over the network. The installer works across all platforms supported by TeX Live. For information on initially downloading the TeX Live, see `http://tug.org/texlive/acquire.html`.

The basic idea of TeX Live installation is 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.

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 do that, use `tlmgr` (reference below) after the initial installation is completely.

The default is `scheme-full`, to install everything, and this is highly recommended.

A.4 REFERENCES

Post-installation configuration, package updates, and much more, are handled through **tlmgr**(1), the TeX Live Manager (`http://tug.org/texlive/tlmgr.html`).

The most up-to-date version of this documentation is on the Internet at `http://tug.org/texlive/doc/install-tl.html`.

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

A.5 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 `=`.

-gui [[=]*module*]

        If no *module* is given starts the `perltk` (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). Same as the `-no-gui` option.

        wizard

                The wizard mode user interface (default on Windows), asking only minimal questions before installing all of TeX Live.

perltk

>   The expert GUI installer, providing access to more options.
>   Can also be invoked on Windows by running `install-tl-advanced.bat`.

The `perltk` and `wizard` modules, and thus also when calling with a bare `-gui` (without *module*), requires the Perl/Tk module (`http://tug.org/texlive/distro.html#perltk`); if Perl/Tk is not available, installation continues in text mode.

-no-gui

>   Use the text mode installer (default except on Windows).

-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).

-repository *url*|*path*

>   Specify the package repository to be used as the source of the installation, either a local directory via `/path/to/directory` or a `file:/` url, or a network location via a `http://` or `ftp://` url. (No other protocols are supported.)
>
>   The default is to pick a mirror automatically, using `http://mirror.ctan.org/systems/texlive/tlnet`; the chosen mirror is used for the entire download. You can use the special argument `ctan` as an abbreviation for this. See `http://ctan.org` for more about CTAN and its mirrors.
>
>   If the repository is on the network, trailing `/` characters and/or trailing `/tlpkg` and `/archive` components are ignored. For example, you could 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 path have to be specified. The list of CTAN mirrors is available at `http://ctan.org/mirrors`.
>
>   If the repository is local, the installation type (compressed or live) is automatically determined, by checking for the presence of a `archive` directory relative to the root. Compressed is preferred if both are available, since it is faster. Here's an example of using a local directory:
>
>   `-repository /local/TL/repository`
>
>   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.
>
>   For backward compatibility and convenience, `--location` and `--repo` are accepted as aliases for this option.

-select-repository

>This option allows manual selection of a mirror from the current list of active CTAN mirrors. This option is supported in all installer modes (text, wizard, perltk), and will also offer to install from local media if available, or from a repository specified on the command line (see above). It's useful when the (default) automatic redirection does not choose a good host for you.

-all-options

>Normally options not relevant to the current platform are not shown (i.e., when running on Unix, Windows-specific options are omitted). Giving this command line option allows configuring settings in the final `texlive.tlpdb` that do not have any immediate effect.

-custom-bin *path*

>If you have built your own set of TeX Live binaries (perhaps because your platform was not supported by TeX Live out of the box), this option allows you to specify the *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 *path* are copied over to `bin/custom/` under your installation directory and this `bin/custom/` directory is what will be added to the path for the post-install actions. (By the way, for information on building TeX Live, see `http://tug.org/texlive/build.html`).

-debug-translation

>In GUI mode, this switch makes `tlmgr` report any missing, or more likely untranslated, messages to standard error. Helpful for translators to see what remains to be done.

-force-platform *platform*

>Instead of auto-detecting the current platform, use *platform*. Binaries for this platform must be present and they must actually be runnable, or installation will fail. `-force-arch` is a synonym.

-help, **--help**, **-?**

>Display this help and exit (on the web via `http://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.

-in-place

>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.

-logfile *file*

>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.

-no-cls

(only for text mode installer) do not clear the screen when entering a new menu (for debugging purposes).

-non-admin

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

--persistent-downloads
--no-persistent-downloads

For network installs, activating this option makes the installer try to set up a persistent connection using the `Net::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`.

-portable

Install for portable use, e.g., on a USB stick. Also selectable from within the perltk and text installers.

-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*

Load the file *profile* and do the installation with no user interaction, that is, a batch (unattended) install.

A *profile* file 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 DEST/tlpkg/texlive.profile. That file can be given as the argument to `-profile` to redo the exact same installation on a different system, for example. Alternatively, you can use a custom profile, most easily created by starting from a generated one and changing values, or an empty file, which will take all the defaults.

Normally a profile has to specify the value `1` for each collection to be installed, even if the scheme is specified. This follows from the logic of the installer in that you can first select a scheme and then change the collections being installed. But for convenience there is an exception to this within profiles: If the profile contains a variable for `selected_scheme` and *no* collection variables at all are defined in the profile, then the collections which the specified scheme requires are installed.

Thus, a line `selected_scheme scheme-medium` together with the definitions of the installation directories (`TEXDIR`, `TEXMFHOME`, `TEXMFLOCAL`, `TEXMFSYSCONFIG`, `TEXMFSYSVAR`) suffices to install the medium scheme with all default options.

-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. You can change the scheme again before the actual installation with the usual menu. The *scheme* argument may optionally have a prefix `scheme-`. The list of supported scheme names depends on what your package repository provides; see the interactive menu list.

-v

> Include verbose debugging messages; repeat for maximum debugging, as in `-v -v`. (Further repeats are accepted but ignored.)

-version, **−version**

> Output version information and exit. If `-v` has also been given the revisions of the used modules are reported, too.

A.6 ENVIRONMENT VARIABLES

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

`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_PREFIX`
`TEXLIVE_INSTALL_TEXMFCONFIG`
`TEXLIVE_INSTALL_TEXMFHOME`
`TEXLIVE_INSTALL_TEXMFLOCAL`
`TEXLIVE_INSTALL_TEXMFSYSCONFIG`
`TEXLIVE_INSTALL_TEXMFSYSVAR`
`TEXLIVE_INSTALL_TEXMFVAR`

> Specify the respective directories.

`NOPERLDOC`

> Don't try to run the `--help` message through `perldoc`.

A.7 AUTHORS AND COPYRIGHT

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

Appendix B tlmgr

B.1 NAME

tlmgr - the TeX Live Manager

B.2 SYNOPSIS

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

B.3 DESCRIPTION

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

The most up-to-date version of this documentation (updated nightly from the development sources) is available at `http://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 `http://tug.org/texlive/doc` for all the TeX Live documentation available.

B.4 EXAMPLES

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

`tlmgr option repository http://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 continuing updates.

`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` *pkg*
> Display detailed information about *pkg*, such as the installation status and description.

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

B.5 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*

> Specifies the package repository from which packages should be installed or updated, overriding the default package repository found in the installation's TeX Live Package Database (a.k.a. the TLPDB, defined entirely in the file `tlpkg/texlive.tlpdb`). The documentation for `install-tl` has more details about this (`http://tug.org/texlive/doc/install-tl.html`).
>
> `--repository` changes the repository location only for the current run; to make a permanent change, use `option repository` (see the Section B.6.13 [option], page 43, action).
>
> For backward compatibility and convenience, `--location` and `--repo` are accepted as aliases for this option.

–**gui** [*action*]

> `tlmgr` has a graphical interface as well as the command line interface. You can give this option, `--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.

–**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).

–**debug-translation**

> In GUI mode, this switch tells `tlmgr` to report any untranslated (or missing) messages to standard error. This can help translators to see what remains to be done.

–**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 58, section below.

–**no-execute-actions**

> Suppress the execution of the execute actions as defined in the tlpsrc files. Documented only for completeness, as this is only useful in debugging.

–package-logfile *file*

> `tlmgr` logs all package actions (install, remove, update, failed updates, failed restores) to a separate log file, by default `TEXMFSYSVAR/web2c/tlmgr.log`. This option allows you to specific a different file for the log.

–pause

> This option makes `tlmgr` wait for user input before exiting. Useful on Windows to avoid disappearing command windows.

–persistent-downloads
–no-persistent-downloads

> For network-based installations, this option (on by default) makes `tlmgr` try to set up a persistent connection (using the `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, `tlmgr` will fall back to using `wget`. To disable these persistent connections, use `--no-persistent-downloads`.

–pin-file

> Change the pinning file location from `TEXMFLOCAL/tlpkg/pinning.txt` (see Section B.10.1 [Pinning], page 54, below). Documented only for completeness, as this is only useful in debugging.

–usermode

> Activates user mode for this run of `tlmgr`; see Section B.7 [USER MODE], page 52, below.

–usertree *dir*

> Uses *dir* for the tree in user mode; see Section B.7 [USER MODE], page 52, below.

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 `http://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 [–clean[=*N*]] [–backupdir *dir*] [–all | *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.

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:

–**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 candidates *pkg*

candidates *pkg*

> Shows the available candidate repositories for package *pkg*. See Section B.10 [MULTIPLE REPOSITORIES], page 54, below.

B.6.5 check [*option*]... [files|depends|executes|runfiles|all]

Executes one (or all) check(s) on the consistency of the installation.

files

> Checks that all files listed in the local TLPDB (`texlive.tlpdb`) are actually present, and lists those missing.

depends

> Lists those packages which occur as dependencies in an installed collections, but are themselves not installed, and those packages that 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.

runfiles

> List those filenames that are occurring more than one time in the runfiles.

Options:

–use-svn

> Use the output of `svn status` instead of listing the files; for checking the TL development repository.

B.6.6 conf [texmf|tlmgr|updmap [–conffile *file*] [–delete] [*key* [*value*]]]

With only `conf`, show general configuration information for TeX Live, including active configuration files, path settings, and more. This is like the `texconfig conf` call, but works on all supported platforms.

With either `conf texmf`, `conf tlmgr`, or `conf updmap` given in addition, shows all key/value pairs (i.e., all settings) as saved in `ROOT/texmf.cnf`, the tlmgr configuration file (see below), or the first found (via kpsewhich) `updmap.cfg` file, respectively.

If *key* is given in addition, shows the value of only that *key* in the respective file. If option *–delete* is also given, the configuration file – it is removed, not just commented out!

If *value* is given in addition, *key* is set to *value* in the respective file. *No error checking is done!*

In all cases the file used can be explicitly specified via the option `--conffile file`, in case one wants to operate on a different file.

Practical application: if the execution of (some or all) system commands via `\write18` was left enabled during installation, you can disable it afterwards:

```
tlmgr conf texmf shell_escape 0
```

A more complicated example: the `TEXMFHOME` tree (see the main TeX Live guide, `http://tug.org/texlive/doc.html`) can be set to multiple directories, but they must be enclosed in braces and separated by commas, so quoting the value to the shell is a good idea. Thus:

```
tlmgr conf texmf TEXMFHOME "{~/texmf,~/texmfbis}"
```

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

B.6.7 dump-tlpdb [–local|–remote]

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 58, section.

Options:

–local

> Dump the local tlpdb.

–remote

> Dump the remote 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.6.8 generate [*option*]... *what*

generate language
generate language.dat
generate language.def
generate language.dat.lua
generate fmtutil

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 `http://tug.org/fonts/fontinstall.html`.

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

The locally-maintained files are `language-local.dat`, `language-local.def`, `language-local.dat.lua`, or `fmtutil-local.cnf`, 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` is no longer read, since `updmap` now supports multiple `updmap.cfg` files. Thus, local additions can and should be put into an `updmap.cfg` file in `TEXMFLOCAL`. The `generate updmap` action no longer exists.)

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

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

where `fmtutil.cnf` uses `#`, `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:

```
#!pdflatex
%!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);
web2c/fmtutil.cnf (and fmtutil-local.cnf);
```

B.6.9 gui

Start the graphical user interface. See **GUI** below.

B.6.10 info [*option...*] [collections|schemes|*pkg...*]

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, installation status, and TeX Live revision number. If *pkg* is not locally installed, searches in the remote installation source.

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 options is given, the installation source will not be used; only locally installed packages, collections, or schemes are listed. (Does not work for listing of packages for now)

–taxonomy
–keyword

–functionality
–characterization

> In addition to the normal data displayed, also display information for given packages from the corresponding taxonomy (or all of them). See Section B.9 [TAXONOMIES], page 53, below for details.

B.6.11 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.7 [USER MODE], page 52, below.

B.6.12 install [*option*]... *pkg*...

Install each *pkg* given on the command line. By default this installs all packages on which the given *pkg*s are dependent, also. Options:

–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).

–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).

–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.

–dry-run

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

–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.

B.6.13 option

option [show]
option showall
option *key* [*value*]

The first form 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 is similar, but also shows options which can be defined but are not currently set to any value.

In the third form, 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    (create formats at installation 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 http://mirror.ctan.org/systems/texlive/tlnet
```

The `install-tl` documentation has more information about the possible values for `repository`. (For backward compatibility, `location` can be used as alternative name 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 what you are doing.

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 perhaps to developers.

The `docfiles` and `srcfiles` options control the installation of their respective files of a package. By default both are enabled (1). This can be disabled (set to 0) if disk space is (very) limited.

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 `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.

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

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

The `--` avoids having the `-1` treated as an option. (`--` 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-like systems to control the generation of links for executables, info files and man pages. See the `path` action for details.

The last three options control behaviour 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.6.14 paper

paper [a4|letter]
[xdvi|pdftex|dvips|dvipdfmx|context|psutils] paper [*papersize*|**–list**]

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.

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.6.15 path [–w32mode=user|admin] [add|remove]

On Unix, merely adds or removes symlinks for binaries, man pages, and info pages in the system directories specified by the respective options (see the Section B.6.13 [option], page 43, description above). Does not change any initialization files, either system or personal.

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 `--w32mode` 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 `--w32mode` is given, this option determines the path to be adjusted.

If the user does not have admin rights, and the option `--w32mode` 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 `--w32mode` is given, it must be **user** and the user path will be adjusted. If a user without admin rights uses the option `--w32mode admin` a warning is issued that the caller does not have enough privileges.

B.6.16 pinning

The `pinning` action manages the pinning file, see Section B.10.1 [Pinning], page 54, 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.6.17 platform list|add|remove *platform...*

B.6.18 platform set *platform*

B.6.19 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`, `xzdec` 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.6.20 postaction [–w32mode=user|admin] [–fileassocmode=1|2] [–all] [install|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.

If the option `--w32mode` is given the value `user`, all actions will only be carried out in the user-accessible parts of the 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 `--w32mode=admin` will not succeed.

`--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`.)

B.6.21 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.6.22 restore [–backupdir *dir*] [–all | *pkg* [*rev*]]

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).

With both *pkg* and *rev*, tries 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.

B.6.23 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:

–no-depends

Do not remove dependent packages.

–no-depends-at-all

See above under **install** (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.

B.6.24 repository

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

This action manages the list of repositories. See Section B.10 [MULTIPLE REPOSITORIES], page 54, below for detailed explanations.

The first form (`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 source from where to initialize a TeX Live Database and lists the contained packages. This can also be an up-to-now not used repository, both locally and remote. If one pass in addition `--with-platforms`, for each package the available platforms (if any) are listed, too.

The third form (`add`) adds a repository (optionally attaching a tag) to the list of repositories. The forth form (`remove`) removes a repository, either by full path/url, or by tag. The last form (`set`) sets the list of repositories to the items given on the command line, not keeping previous settings

In all cases, one of the repositories must be tagged as `main`; otherwise, all operations will fail!

B.6.25 search [*option...*] *what*

B.6.25.1 search [*option...*] –file *what*

B.6.25.2 search [*option...*] –taxonomy *what*

B.6.25.3 search [*option...*] –keyword *what*

B.6.25.4 search [*option...*] –functionality *what*

B.6.25.5 search [*option...*] –characterization *what*

B.6.25.6 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 regular expression.

Options:

–global

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

–word

> Restrict the search 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).

–list

> If a search for any (or all) taxonomies is done, by specifying one of the taxonomy options below, then instead of searching for packages, list the entire corresponding taxonomy (or all of them). See Section B.9 [TAXONOMIES], page 53, below.

Other search options are selected by specifying one of the following:

–file

> List all filenames containing *what*.

–taxonomy
–keyword

–functionality
–characterization

> Search in the corresponding taxonomy (or all) instead of the package descriptions. See Section B.9 [TAXONOMIES], page 53, below.

–all

> Search for package names, descriptions, and taxonomies, but not files.

B.6.26 uninstall

Uninstalls the entire TeX Live installation. Options:

–force

> Do not ask for confirmation, remove immediately.

B.6.27 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. Thus, if updates to `tlmgr` itself are present, this will simply give 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`.

> 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`.

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

–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:

```
tlmgr update --self        # update infrastructure only
tlmgr update --self --all  # update infrastructure and all packages
tlmgr update --force --all # update all packages but *not* infrastructure
                           # ... 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, `tlmgr update --list` is the same as `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,

```
tlmgr update --all --exclude a2ping
```

will not update `a2ping`, `a2ping.i386-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.

–no-auto-remove [*pkg*]...

By default, `tlmgr` tries to remove packages 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 recommend.

–**no-auto-install** [*pkg*]...

> Under normal circumstances `tlmgr` will install packages which are new on the server, as described above under `--all`. This option prevents any such automatic installation, either for all packages (with `--all`), or the given *pkg* names.
>
> Furthermore, after the `tlmgr` run using this has finished, the packages that would have been auto-installed *will be considered as forcibly removed*. So, if `foobar` is the only new package on the server, then
>
> ```
> tlmgr update --all --no-auto-install
> ```
>
> is equivalent to
>
> ```
> tlmgr update --all
> tlmgr remove --force foobar
> ```

–**reinstall-forcibly-removed**

> Under normal circumstances `tlmgr` will not install packages that have been forcibly removed by the user; that is, removed with `remove --force`, or whose installation was prohibited by `--no-auto-install` during an earlier update.
>
> This option makes `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:
>
> ```
> tlmgr update --reinstall-forcibly-removed --all
> ```

–**backup** and –**backupdir** *directory*

> These two options control the creation of backups of packages *before* updating; that is, backup of packages as currently installed. If neither of these options are given, no backup package will be saved. If `--backupdir` is given and specifies a writable directory then a backup will be made in that location. If only `--backup` is given, then a backup will be made to the directory previously set via the `option` action (see below). If both are given then a backup will be made to the specified *directory*.
>
> You can set options via the `option` action to automatically create backups for all packages, and/or keep only a certain number of backups. Please see the `option` action for details.
>
> `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 `--backup` option is to allow you to save a persistent backup in case the actual *content* of the update causes problems, e.g., introduces an incompatibility.
>
> The `restore` action explains how to restore from a backup.

–**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 **install** (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.

B.7 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`, `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.7.1 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.

B.7.2 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.7.3 user mode generate; option; paper

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

B.8 CONFIGURATION FILE FOR TLMGR

A small subset of the command line options can be set in a config file for `tlmgr` which resides in `TEXMFCONFIG/tlmgr/config`. By default, the config file is in `~/.texliveYYYY/texmf-config/tlmgr/config` (replacing `YYYY` with the year of your TeX Live installation). This is *not* `TEXMFSYSVAR`, so that the file is specific to a single user.

In this file, empty lines and lines starting with # are ignored. All other lines must look like

```
key = value
```

where the allowed keys are `gui-expertmode` (value 0 or 1), `persistent-downloads` (value 0 or 1), `auto-remove` (value 0 or 1), and `gui-lang` (value like in the command line option).

`persistent-downloads`, `gui-lang`, and `auto-remove` correspond to the respective command line options of the same name. `gui-expertmode` switches between the full GUI and a simplified GUI with only the important and mostly used settings.

B.9 TAXONOMIES

tlmgr allows searching and listing of various categorizations, which we call *taxonomies*, as provided by an enhanced TeX Catalogue (available for testing at `http://az.ctan.org`). This is useful when, for example, you don't know a specific package name but have an idea of the functionality you need; or when you want to see all packages relating to a given area.

There are three different taxonomies, specified by the following options:

`--keyword`
> The keywords, as specified at `http://az.ctan.org/keyword`.

`--functionality`
> The "by-topic" categorization created by J\"urgen Fenn, as specified at `http://az.ctan.org/characterization/by-function`.

```
--characterization
          Both the primary and secondary functionalities, as specified at http://az.
          ctan.org/characterization/choose_dimen.

--taxonomy
          Operate on all the taxonomies.
```

The taxonomies are updated nightly and stored within TeX Live, so Internet access is not required to search them.

Examples:

```
tlmgr search --taxonomy exercise       # check all taxonomies for "exercise"
tlmgr search --taxonomy --word table   # check for "table" on its own
tlmgr search --list --keyword          # dump entire keyword taxonomy
tlmgr show --taxonomy pdftex           # show pdftex package information,
                                       #   including all taxonomy entries
```

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 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://tlcontrib.metatex.org`, maintained by Taco Hoekwater et al.), with the tag `tlcontrib`:

```
$ tlmgr repository add http://tlcontrib.metatex.org/2012 tlcontrib
```

Check the repository list again:

```
$ tlmgr repository list
List of repositories (with tags if set):
    http://tlcontrib.metatex.org/2012 (tlcontrib)
    /var/www/norbert/tlnet (main)
```

Now we specify a pinning entry to get the package `context` from `tlcontrib`:

```
$ tlmgr pinning add tlcontrib context
```

Check that we can find `context`:

```
$ tlmgr show context
tlmgr: package repositories:
...
package:    context
repository: tlcontrib/26867
...
```

  - install `context`:

```
$ tlmgr install context
tlmgr: package repositories:
...
[1/1,  ??:??/??:??] install: context @tlcontrib [
```

In the output here you can see that the `context` 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 context  # remove just context
 $ tlmgr pinning remove tlcontrib --all    # take nothing from tlcontrib
```

A summary of the `tlmgr pinning` actions is given above.

B.11 GUI FOR TLMGR

The graphical user interface for `tlmgr` needs Perl/Tk to be installed. For Windows the necessary modules are shipped within TeX Live, for all other (i.e., Unix-based) systems Perl/Tk (as well as Perl of course) has to be installed. `http://tug.org/texlive/distro.html#perltk` has a list of invocations for some distros.

When started with `tlmgr gui` 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 `tlmgr` menu item.

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

B.11.1 Main display

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.

Status

>   Select whether to show all packages (the default), only those installed, only those *not* installed, or only those with update available.

Category

>   Select which categories are shown: packages, collections, and/or schemes. These are briefly explained in the Section B.3 [DESCRIPTION], page 35, section above.

Match

>   Select packages matching for a specific pattern. By default, this uses the same algorithm as `tlmgr search`, i.e., searches everything: descriptions, taxonomies, and/or filenames. You can also select any subset for searching.

Selection

>   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.

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 are 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.

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.8 [CONFIGURATION FILE FOR TLMGR], page 53, 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 `http://tug.org/texlive/doc.html`) and the usual "About" box.

B.12 MACHINE-READABLE OUTPUT

With the `--machine-readable` option, `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.6.27 [update], page 49, Section B.6.12 [install], page 42, and Section B.6.13 [option], page 43, actions.

B.12.1 Machine-readable `update` and `install` output

The output format is as follows:

```
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
```

The header section currently has two fields: `location-url` (the repository source from which updates are being drawn), and `total-bytes` (the total number of bytes to be downloaded).

The *localrev* and *serverrev* fields for each package are the revision numbers in the local installation and server repository, respectively. The *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 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.

size

> The size in bytes of the package on the server. The sum of all the package sizes is given in the `total-bytes` header field mentioned above.

runtime

> The run time since start of installations or updates.

esttot

> The estimated total time.

B.12.2 Machine-readable `option` output

The output format is as follows:

```
key "\t" value
```

If a value is not saved in the database the string (`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.

B.13 AUTHORS AND COPYRIGHT

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

Index

C

D

E

F

G

I