Redistributing TEX and friends

Norbert Preining

Abstract

Nowadays most TEX installations are based on TEX Live. TUG provides a platform-independent installer which can be used on many different platforms. But operating system distributors, such as Debian and Red Hat normally integrate TEX Live into their own packaging infrastructure.

Based on years of experience in packaging TEX Live for Debian, as well as upstream development, we give here a short introduction to the TEX Live ecosystem, list important files in need of special care when redistributing TEX Live, and give advice and warnings.

1 Introduction

The TEX environment has grown slowly but steadily into a huge collection of programs, fonts, macros, documentation, and more. TEX Live currently ships over 3 GB in more than 2500 different TEX Live "packages", most of which are installed into TEX Live from CTAN [1]. Since teTEX development and support stopped several years ago, TEX Live has become the main TEX distribution on Unix, including Mac OS X (MacTEX is exactly TEX Live plus a few Mac-specific additions); it is also gaining on Windows (where MiKTEX is still strong).

Integrating TEX Live into any full operating system distribution is a non-trivial task due to the large number of post-installation tasks that have to be performed. Although over the last years the quality of packages has improved, the TEX Live development list still often gets bug reports that stem from incorrect packaging.

In the following we will give an overview of the structure of TEX Live and a list of important and special configuration files. Furthermore, based on the experience of packaging TEX Live over many years, we will give some advice and examples of best practices. Although the author maintains TEX Live for Debian, the information in this article is not targeted specifically for Debian, but at any distribution that redistributes TEX Live in one way or another.

The layout of the article is as follows: We will first give an overview of the structure of the TEX Live ecosystem. After that we discuss stacked versus non-stacked configuration files, followed by a discussion of the most important configuration files that need special handling. Finally, we collect some ideas concerning approaches to packaging TEX Live found in distributions.

2 Structure of TEX Live

TEX Live currently ships something like 130,000 files. To make this vast amount of material easier to handle we have introduced a hierarchical organization.

Schemes form the topmost level, with a dozen or so schemes defined. The default is `scheme-full`, which installs everything; at the other extreme is `scheme-minimal`, which installs only enough to run plain TEX. Schemes contain overlapping content; e.g., clearly everything in `scheme-minimal` is also contained in `scheme-full`.

Collections form the middle layer, with currently 45 collections. Each collection contains related (to some degree) packages. An example here is `collection-latex`. In contrast to the schemes, the collections form a mathematical partition of the content, that is, non-overlapping: every package is in exactly one collection.

Packages form the bottom layer, with currently around 2500 packages. As mentioned, most packages relate to an item available through CTAN. Examples are `pdftex` and `beamer`. A given file is in exactly one package.

2.1 TEX Live database

The TEX Live database, in short `tlpdb`, is a file usually located under the main installation's root in `tlpg/texlive.tlpdb`. It is a simple text file where information is line based, and blocks (stanzas) are separated by blank lines. The structure is very similar to a Debian `Packages` file. Each stanza describes a package:

```
name beamer
...

name pdftex
...
```

Each non-empty line is either a `key value` pair or a file name, as we will see.

2.2 Package description

Each package contains various information: its name, a revision number, dependencies (`depends`), special things to be done when the package is installed (`execute`), and lists of files in three categories: runtime files (`runfiles`), binary (executable) files including scripts (`binfiles`), and documentation files (`docfiles`). The package description is also enriched with information obtained from the TEX Catalogue. A more or less complete example for a package stanza can be found in fig. 1.

```
name ascii-font
category Package
revision 29989
shortdesc Use the ASCII "font" in LaTeX.
longdesc The package provides glyph and font ...
longdesc ... and R.W.D. Nickalls.
execute addMap ascii.map
containersize 48984
containermd5 8e922125b755694d21b45e9644265611
doccontainersize 552
doccontainermd5 7b0c7918dadaca7665f8d1bd61677254
docfiles size=1
 texmf-dist/doc/fonts/ascii-font/README.TEXLIVE
srccontainersize 4444
srccontainermd5 82f12b5dbe4107bada602b7f0dcb5561
srcfiles size=5
 texmf-dist/source/fonts/ascii-font/ascii.dtx
 texmf-dist/source/fonts/ascii-font/ascii.ins
runfiles size=17
 texmf-dist/fonts/map/dvips/ascii-font/ascii.map
 texmf-dist/fonts/tfm/public/ascii-font/ASCII.tfm
 texmf-dist/fonts/type1/public/ascii-font/ASCII.pfb
 texmf-dist/tex/latex/ascii-font/ascii.sty
catalogue-ctan /fonts/ascii
catalogue-date 2013-04-15 01:42:14 +0200
catalogue-license lppl
catalogue-version 2.0
```

Figure 1: Stanza for the `ascii-font` package

We will come back to this example later, after discussing the various configuration files.

3 Types of configuration files

Most of the files in a TEX system are normal input files. These files are searched for using the well-known Kpathsea library. Normally, only the first-found file is read (details of the file search algorithm are in the Kpathsea manual [3]). This is the normal case, and we will refer to it henceforth as the *non-stacked* case.

In contrast, a few configuration files are read in a *stacked* manner, where *all* files found by Kpathsea are read and evaluated, not just the first.

The difference can be seen in a TEX Live installation by comparing the `kpsewhich updmap.cfg` output to that of `kpsewhich -all updmap.cfg`. On my system (which is installed in `/tl/2013` and is a bit unusual with respect to `texmf-local`) I get:

```
$ kpsewhich updmap.cfg
/tl/2013/texmf-config/web2c/updmap.cfg
$ kpsewhich -all updmap.cfg
/tl/2013/texmf-config/web2c/updmap.cfg
/usr/local/share/texmf/web2c/updmap.cfg
/tl/2013/texmf-dist/web2c/updmap.cfg
```

In the non-stacked case only the first file would be read; in the stacked case, all of them.

Not many files are treated in a stacked way. In the next section we will discuss the most important configuration files and mention for each whether it is read in a stacked or non-stacked way.

4 Important configuration files

The TEX Live configuration files discussed here are the most important, especially for distributors, as they need special attention. Other configuration files (there are plenty more) can be treated transparently, as they should generally work without any changes.

The configuration files we will discuss are:

`texmf.cnf` Central configuration file for path searching and parameters of the engines.

`updmap.cfg` Configuration file for font embedding from which configurations for driver programs are produced.

`fmtutil.cnf` TEX formats (and METAFONT bases).

`language.dat` Several files controlling the inclusion of hyphenation patterns in format dumps.

For each of these we give advice on what distributors can (should?) change and how they can be handled.

4.1 `texmf.cnf`

The `texmf.cnf` file defines the available trees, among many other things. It has always been treated as a *stacked* configuration file — all the `texmf.cnf` files found are evaluated. This feature is used in the native `install-tl` to adjust settings via a file `texmf.cnf` at the root of the TEX Live installation.

By default the following trees are defined and used, where `R` is the root of the installation:

`TEXMFDIST` files from TEX Live	`R/texmf-dist`
`TEXMFHOME` user tree	`~/texmf`
`TEXMFLOCAL` site-wide additions	`R/../texmf-local`
`TEXMFSYSVAR` cached data	`R/texmf-var`
`TEXMFSYSCONFIG` config. data	`R/texmf-config`
`TEXMFVAR` per-user cached data	`~/.texlive2013/texmf-var`
`TEXMFCONFIG` per-user modified configuration data	`~/.texlive2013/texmf-config`
`VARTEXFONTS` location of generated fonts	`TEXMFVAR/fonts`

In recent years, when packaging for Debian I haven't needed to change anything outside of these path definitions. In particular, distributors might want to change the definition of `TEXMFSYSCONFIG`. In Debian, we change that to `/etc/texmf` in accordance with our policies.

Another possible adjustment is adding an additional tree. In Debian, we ship TEX Live in `/usr/share/texlive` and add a tree called `TEXMFDEBIAN` in `/usr/share/texmf`, searched before `TEXMFDIST`.

To effect such changes, distributors can either patch the main `texmf.cnf` in `texmf-dist/web2c`

or add another `texmf.cnf` in one of the searchable trees. Of course, one cannot change the location of, say, `TEXMFSYSCONFIG` to a different path in a `texmf.cnf` file *within* the new location. So in Debian we patch the main configuration file to adjust *only* `TEXMFSYSCONFIG`, and add all other changes to `/etc/texmf/web2c/texmf.cnf`.

4.2 `updmap.cfg`

The `updmap.cfg` file has probably caused the most grief, so we will go to great length in the explanations.

Many of the fonts shipped in TeX Live are PostScript Type 1 fonts. TeX itself does not know anything about these fonts, and only uses the metrics (`.tfm`). Output drivers, on the other hand, need to know how the metrics are mapped to external fonts. Some notable output drivers:

`pdftex` The TeX engine with PDF output. Since producing PDF clearly needs the actual fonts, `pdftex` is also an output driver.

`dvips` A classic output driver converting `.dvi` (DeVice Independent) files to PostScript. Again, the fonts have to be embedded.

`(x)dvipdfm(x)` The family of dvi-to-pdf converters. These programs support direct translation from DVI to PDF. XeTeX uses one of these in the background. Japanese users often use `dvipdfmx`, since it has good support for Japanese fonts.

`(p)xdvi` Display programs, of course need access to the fonts. `pxdvi` is `xdvi` patched for Japanese support.

Unfortunately, different drivers need the font mapping in different formats. Here is where `updmap` comes into play: It reads a list of specifications and creates configuration files for the above programs, in the necessary formats.

4.2.1 Different layers of configuration

The files generated by `updmap` have a long chain of provenance:

- A "font map definition" maps a `.tfm` file name to an external font with optional transformations.
- A "font map file" collects font map definitions; normally there is one font map file per package, collecting all fonts in that package.
- An "`updmap` config file" lists options and font map files.
- The "output driver configuration files" are read by the output drivers; these files are generated by `updmap`.

4.2.2 Configuration of fonts in `updmap.cfg`

`updmap`'s central configuration file is `updmap.cfg`. In former times, only the first one found by Kpathsea was used, but now all of them are read (see below). Each `updmap.cfg` file can contain either empty lines, comment lines starting with the comment char `#`, or one of the following settings, in the format `key value`:

`dvipsPreferOutline true` or `false`; whether `dvips` uses bitmaps or outlines, where possible.

`dvipsDownloadBase35 true` or `false`; whether `dvips` embeds the standard 35 PostScript fonts.

`pdftexDownloadBase14 true` or `false`; whether `pdftex` embeds the standard 14 PDF fonts.

`pxdviUse true` or `false`; whether `updmap` controls `pxdvi`'s maps.

`kanjiEmbed`,`kanjiVariant` arbitrary strings, controlling kanji font embedding

`LW35` `URWkb`, `URW`, `ADOBEkb`, `ADOBE`; file naming scheme assumed for the base PostScript fonts.

map directives One of `Map foo.map`, `MixedMap bar.map`, or `KanjiMap baz.map`. `Map` is used for fonts that are available only in PostScript format; `MixedMap` for fonts where Metafont and PostScript variants are present, and `KanjiMap` for special kanji support (see [5]).

4.2.3 Operation mode

As of TeX Live 2013, `updmap` reads all `updmap.cfg` files found, i.e., all the files given by `kpsewhich -all updmap.cfg`, in contrast to the former method of only reading the first one found.

We made this change for several reasons. First, it supports having the font map configuration in the same tree as the fonts themselves. Before, the activation of a map file did not survive when (re)installing a new release of TeX Live. Now, if for example `TEXMFLOCAL` contains local fonts, and they are listed in `TEXMFLOCAL/web2c/updmap.cfg`, they will automatically be picked up. A second reason is to support users without write permission to the system installation. This way, they can manage their fonts without needing a copy of the system's `updmap.cfg`.

More specifics, such as enabling and disabling of maps, can be found in the manual page of `updmap` and a blog post [2].

4.2.4 Recommendations for distributors

Distributors must be aware that changing the set of available fonts requires a change to one of the `updmap.cfg` files, followed by running `updmap-sys`. Otherwise, the fonts will not be available to users, even though they are present in the system. Also, distributors should *not* ship the `updmap.cfg` file included in TeX Live, since it is only valid for a full installation of TeX Live. (The TeX Live installer

itself does not install this file, but generates it from the set of installed packages.)

Since `updmap.cfg` is read in a stacked manner, changes can be localized to the tree where fonts are installed. In Debian we have one `updmap.cfg` for the TeX Live packages in `/usr/share/texlive/texmf-dist/web2c/updmap.cfg`, and one for additional font packages with files in `TEXMFDEBIAN`.

4.3 `fmtutil.cnf`

The configuration files discussed so far have been read in a stacked way; the following files are all non-stacked. To repeat that important difference, only one instance of the following files will be used, namely the one that is returned by a normal `kpsewhich` call.

`fmtutil.cnf` is the main configuration file for the `fmtutil` program, which generates format dumps for the various engines. Thus, a change in available formats needs to change `fmtutil.cnf`, and then call `fmtutil-sys`.

Fortunately, it is rare that a user wants to create his own format dumps (and such users can take care of themselves); so distributors need only make sure that the configuration file stays properly updated.

4.4 `language.dat` family

The last group of configuration files relates to the definition of hyphenation patterns. Many engines load hyphenation patterns for different languages at format dump time (see above), and proper hyphenation is possible with only those languages. These files are:

`language.dat` for LaTeX-based formats

`language.def` for ε-TeX-based plain formats

`language.dat.lua` for LuaTeX-based formats

The first two files are loaded at format dump time, thus a change in the available hyphenation patterns needs to (again) trigger a call to `fmtutil-sys`, best in combination with the `--byhyphen` command line option to specify explicitly the location of the hyphenation file.

The last of the three is easier, since LuaTeX loads the patterns at runtime. So no action on the side of the distributors is necessary.

5 Gluing it together

5.1 Execute statements

Many times above I have written 'change in availability'. But how can a distributor detect such a change? The answer lies in the `execute` statements in the package stanzas, as shown in fig. 1. There are three different execute actions: one for font maps, one for formats, and one for hyphenation patterns.

5.1.1 Font map execute action

Activating a font can happen in three different ways, trivially corresponding to the three different map types in `updmap.cfg`:

`execute addMap` ⟨*mapname*⟩
: Add a line '`Map` ⟨*mapname*⟩'.

`execute addMixedMap` ⟨*mapname*⟩
: Add a line '`MixedMap` ⟨*mapname*⟩'.

`execute addKanjiMap` ⟨*mapname*⟩
: Add a line '`KanjiMap` ⟨*mapname*⟩'.

For distributors, this means that part of creating the TeX packages for distribution is determining the maps to be activated from the `tlpdb`, and adding the respective lines to the appropriate `updmap.cfg` file. The semantic differences between the three invocations are explained in the `updmap` documentation.

5.1.2 Format execute action

The information involved in defining a format is a bit more complex than for font maps. Each execute statement contains again a list of `key=value` pairs, all on the same line in the `tlpdb`. The possible keys are `name`, `engine`, `patterns`, and `options`. A typical line (breaks are due to *TUGboat*):

```
execute AddFormat name=pdflatex engine=pdftex
patterns=language.dat
options="-translate-file=cp227.tcx *pdflatex.ini"
```

The value of `options` often contains spaces and thus needs to be quoted.

The meaning of the above line is that a line

`name engine patterns options`

with the respective values should be added to the `fmtutil.cnf` file (without any quotes). In the above case, that would be:

```
pdflatex pdftex language.dat
-translate-file=cp227.tcx *pdflatex.ini
```

5.1.3 Hyphenation execute action

The most complicated execute statement regards the activation of hyphenation patterns. As in the previous case, it is a line of `key=value` pairs. The possible keys this time are `name`, `synonyms`, `lefthyphenmin`, `righthyphenmin`, `file`, `file_patterns`, and `file_exceptions`. How to generate the three language definition files (`.dat`, `.def`, `.dat.lua`) from this information is beyond the scope of this article; there are functions available in the Perl modules distributed with TeX Live (in `tlpkg/TeXLive`).

5.2 Distribution paradigms

When it comes to distributing such a huge piece of software, several options have been used. The first

question is perhaps *if and how* to split the full TEX Live before repackaging it for a distribution. This gives rise to choosing one of the following paradigms:

all-or-nothing all of TEX Live is distributed as one distribution package

collection-splitting one distribution package per TEX Live collection

package-splitting one distribution package per TEX Live package

mixed-mode overlapping/ad hoc splitting

The *all-or-nothing* approach has the advantage that, in principle, no changes to the various config files are needed. Just ship them as they should be and that's it. Unfortunately, this is no longer the case as soon as there are fonts shipped independently from TEX Live in the distribution. Even worse, downloading a few Gb for any update will not make the users of your distribution happy. I don't know of any distribution using this method.

The *collection-splitting* approach converts TEX Live collections into distribution packages. This approach has many advantages: first, since the content of collections do not overlap, there will be no file conflict (double inclusion) which is a basic requirement for package managers. Furthermore, TEX Live collections try to group related packages together, so users can potentially eliminate collections not of interest. Finally, the number of distribution packages is not too big. On the negative side, splitting by collection requires a bit more work on the packaging side. Debian and its derivatives (such as Ubuntu) use this approach.

The *package-splitting* approach converts each TEX Live package to one distribution package. While this is conceptually the cleanest approach, and allows for fine-grained installations, it requires a near-fully automatic packaging system due to the huge number of packages. Having thousands of distribution packages itself might be regarded as a disadvantage. Furthermore, since we do not track inter-package dependencies in TEX Live, dependencies between distribution packages will be incomplete. Distributions using this paradigm include Fedora and SuSE.

Finally, as I understand it, the *mixed-mode* paradigm is used in some BSD packaging, but I don't know the details and so cannot comment on advantages or disadvantages.

6 Closing

I want to close with some warnings and common pitfalls.

I've never used TEX Live and I don't know what TEX does, but I package it! — it sounds crazy, but we have heard from people who want to package TEX without the slightest knowledge. Just say no.

Improper configuration file handling — by far the biggest problem, and the reason I wrote this article. A common error is shipping the TEX Live `updmap.cfg` instead of generating its content based on the fonts actually installed.

What is upstream? — since TEX Live has (approximately) one release per year and daily updates, it is rather a moving target. Build scripts that require a stable target (such as some BSD ports) need to take extra care.

Binaries and sources — we almost never update compiled binaries after a release, but our development sources are changing continually. Thus, it's a mistake to base distribution binaries on them.

Shipping `tlmgr` — distributions have their own package manager, thus subsuming the most important part of `tlmgr` functionality. Even if users are crying for it, `tlmgr` should not be used to update packages (also not by root). The only reasonable approach is to ship `tlmgr` working in *user mode* only, where it manages `TEXMFHOME`.

In addition to the above, I highly recommend creating a working installation of TEX Live and actually use it yourself; and to learn Perl, since most of the functionality of our installer and `tlmgr` are implemented in Perl modules and available in the TEX Live distribution; and finally, to contact us — we have a designated mailing list for distributors [4].

References

[1] CTAN (Comprehensive TEX Archive Network). `http://ctan.org`.

[2] Internals of TEX Live 2: multi-updmap. `http://www.preining.info/blog/2013/07/internals-of-tex-live-2-multi-updmap/`.

[3] Kpathsea manual. `http://tug.org/kpathsea`.

[4] TEX Live distributors mailing list. `http://lists.tug.org/tldistro`.

[5] Updmap and Kanji embedding in TEX Live. `http://tug.org/texlive/updmap-kanji.html`.

◇ Norbert Preining
Japan Advanced Institute of Science and Technology
Nomi, Ishikawa, Japan
`norbert (at) preining dot info`
`http://tug.org/texlive`