The LaTeX release workflow and the LaTeX dev formats

Frank Mittelbach, LaTeX Project

Abstract

How do you prevent creating banana software (i.e., software that gets ripe at the customer site)? By proper testing! But this is anything but easy.

The paper will give an overview of the efforts made by the LaTeX Project Team over the years to provide high-quality software and explains the changes that we have made this summer to improve the situation further.

Contents

Some history

The LaTeX Project team's attempts to provide reliable high-quality software can be traced back to the first days when we took over LaTeX 2.09 from Leslie Lamport for maintenance and started to work on producing a new major version of LaTeX which came into existence around 1993 under the name of LaTeX 2_ε. In its core this is still the version you are using today, albeit these days with many extensions and additional kernel features.

While LaTeX 2_ε looked fairly similar on the user interface level (to allow for easy transition), under the hood it used completely different internal concepts in some parts of the software and was therefore quite different in many areas from the LaTeX 2.09 version. After all, the goal of the new system was to overcome (most) limitations and deficiencies of the original program that had surfaced since its introduction in 1986.

Executing such major software changes was and is a daunting task, especially when a huge user base relies on the software to continue to function seamlessly with old and new documents (and preferably produce identical results).

To have a fighting chance of success, we came up with the idea of a regression test suite for LaTeX where certified results from document runs could be used to automatically check that changes made to the internal code base did not affect the behavior of LaTeX document level interfaces, etc.[1] A good overview of the features and mechanisms of the early regression test suite can be found in [1].

Around 1992 we then initiated several volunteer projects [2], one of which was helping to build a base set of test files which would test all interfaces, and additionally provide unit tests for known bugs and issues that we intended to fix.

It was largely due to Daniel Flipo (coordinator back then) and the volunteers who worked on this initial test set that the introduction of LaTeX 2_ε turned out to be quite successful. Even though we experienced some problems and also some level of push back (as in "Why do we need a new system which is so much bigger and contains all these fonts with accented characters that nobody is ever going to need . . . "), on the whole the system was favorably received and after a round of smaller maintenance releases that fixed overlooked issues and added a few more missing bits and pieces, the kernel settled to a stable state with little update or extension. Instead, further development moved into the package area, for which the new kernel provided a better basis than the old.

The growing release support infrastructure

With each bug that was found and any new feature that got added, the test suite grew. These days it contains roughly 700 test documents. There are about 400 for the core of LaTeX and another 300 for the packages that form the required set, i.e., `amsmath`, `graphics` and `tools`, but excluding `babel`, which has its own test suite and release cycle.

At the beginning the support scripts to automatically run the tests were Unix-based (mainly a huge `Makefile` plus a few `bash` scripts). We then switched over to a Windows-based system and finally, when LuaTeX became generally available in the distributions, rewrote the whole system in Lua. Thanks mainly to Joseph Wright, Will Robertson and others this `l3build` program is now a very flexible and sophisticated tool that is not only capable of running the test suite for our own LaTeX work, but also able to automate the whole release cycle up to and including automatic uploads to CTAN [3, 4].

As its actions are configured through a simple but powerful configuration file that you maintain

[1] Nothing especially spectacular these days, but around 1990 it was a rather uncommon approach.

in your package source tree, it is perfectly capable of supporting any sort of package development in the TEX world (it doesn't have to be LATEX). So if you are a developer and haven't seen it yet, try it out; it will most certainly make your life simpler and through its automation, more reliable.

The typical release workflow for LATEX

With `l3build`, our typical release work flow these days looks roughly like this:

1. Development phase

– Make some changes (bug fixes or minor extensions) to the LATEX kernel or to core packages.

– Write some new test files to cover the change and the expected behavior.

2.a Testing phase (run `l3build check`)

– This way we immediately see if the changes break anything.

– All tests are executed with pdfTEX, XETEX, LuaTEX and for a certain subset also with pTEX and upTEX.

– This means that running `l3build` with the `check` target executes more than 2000 test documents for the LATEX 2_ε distribution, which even on a reasonably fast machine requires some patience.

2.b Testing phase (do a `texmf-dist` search)

– If we had to modify internal kernel commands we do a sweep over the whole TEX Live distribution tree to check if those commands have been used or (often more importantly) modified by other packages and whether or not our change will conflict with that usage.

– If so, we analyze the situation further and inform the package authors to coordinate any necessary updates.

– Depending on the analysis, we may also conclude that we need to revert our change or implement it differently.

2.c Testing phase (asking for user testing)

– This requires manually installing a prerelease of the new kernel locally and thus is unfortunately both somewhat time consuming and requiring knowledge that is not so easy to come by these days.

– As a result we had little to no feedback in the last years from this step.

3. Everything is finally "in the Green"

– ... or so everybody believes.

4.a Upload phase (run `l3build ctan`)

– This target reruns all checks with all engines, processes all documentation and if this succeeds without any errors, collects the result in a `.zip` for upload to CTAN.

4.b Upload phase (run `l3build upload`)

– Based on preconfigured information this target automatically uploads the `.zip` file together with the necessary metadata to CTAN. For some pieces of information, such as installation instructions to the CTAN team, we are prompted, as they change from upload to upload.

4.c Upload phase (reaching the distributions)

– Once installed on CTAN the kernel and packages move into the TEX distributions, e.g., TEX Live and MikTEX, and then, depending on the update policy chosen by the end user, show up on the user machines either automatically or through a manually initiated update process. And then ...

BOOM!

– Thousands of users use (a.k.a. "test") the changes and a few encounter issues, due to dependencies our test suite hasn't signaled, packages we have overlooked, code or packages that are not on CTAN, etc.

5. Urgent patch phase

– The problem is that with the user base measuring in millions[2] even a rate of 0.001% of users being affected by some issue translates into a noticeable number of users with problems.

– Thus, even a single issue with some nearly-unused package may need urgent correction (and it takes a few days from producing a patch to getting it into end user hands). As a consequence this is usually a phase of hectic activity and we have seen in recent releases more than one patch needing to be provided in quick succession — the worst case was three in 2016.

[2] Nobody knows for sure how many active LATEX users are out there as there is no easy way to measure this. Downloads of TEX Live or from CTAN, for example, are done through a large network of mirrors and the download numbers per mirror are unknown. But there are somewhat between six and ten thousand hits on the LATEX project web site per day, most of which look at the "get" or "about" pages, i.e., are most likely new users. Another indication is that visitor numbers grow substantially at the start of university terms. This would mean more than two million prospective new users hitting the site per year.

Reasons for failure

What are the reasons that despite extensive regression testing we often end up with patch releases? They are largely due to the ineffectiveness of steps 2.b and 2.c in the above sequence.

The regression test suite we run in step 2.a ensures that our official interfaces are all working correctly and any bugs we have fixed in the past do not suddenly reappear. In fact on several occasions it has saved us from major blunders by stopping us from distributing "harmless changes that couldn't by any chance produce problems" but did after all — often in, on the surface, unrelated places.

Finding all these dependencies up front

However, even a huge test suite can't find and test for all kinds of possible dependencies in several thousand contributed packages and millions of user documents. This problem is increased by the fact that the LaTeX code was written for a very constrained engine and the kernel is therefore very much tailored and streamlined, saving token space whenever possible.[3] As a result there aren't many interfaces where third-party packages can officially hook into kernel functionality, so it isn't surprising that there is nearly no internal LaTeX command that hasn't been (mis)used in one way or another by some package out there.

This is the reason for the importance of step 2.b, but since this is largely a manual effort it is easy to miss cases or fool oneself into believing that no one could possibly have altered *this or that* internal command in a package — in the end somebody usually did after all.

The missing maintainer problem

The other issue with step 2.b is that these days there are unfortunately many packages in use where the original author is no longer reachable, because he or she has moved on. Their packages are on CTAN and in the distributions but the maintainer information is no longer correct. As long as everything works that is not necessarily a problem, but the moment something breaks it can be quite hard or even impossible to find the person and even if the search is successful it may turn out that they no longer have any interest in their work which they did years ago.

A recent prominent example of this problem is the package `tabu` which implements a nice user syntax on top of `array`, `tabularx` and `longtable`. That package was abandoned by its author around 2011 but people continued to use it despite a few unfixed (minor) bugs, because it does implement a number of nice ideas.

Unfortunately, its code hacks in rather bad ways into the kernel internals in places that should never have been altered by other packages. So when we had to fix a color leakage problem in tabular cells in the core kernel commands by adding color safe groups that then broke the package for good. Without a maintainer who was willing to spend the necessary time to unravel these hacks in the package code, the package remained broken when the 2018 kernel got released.[4]

The biggest problem resulting from this was that `doxygen`, the de-facto open source standard for producing annotated C++ code documentation was making heavy use of the `tabu` package when producing LaTeX-based PDF documents. As a result their toolset was initially unable to use the current LaTeX release. We recently resolved this for them by providing a dedicated rollback of the involved kernel fixes to be used within their workflow (i.e., reintroducing the kernel bug so that a special version of `doxygen-tabu` could be used as part of their documentation tools). This is clearly far from a perfect solution, so we hope that a new maintainer for `tabu` will eventually step forward so that this rollback can be removed again.

No or not enough third-party testing

However, we believe that the most important factor for ending up with patch level releases was in step 2.c: the insufficient public testing of the release prior to its move into the main distributions.

In essence, the effort needed from users was simply too high and the setup too complicated, so only a very small number of people participated. Testing was therefore neither sufficient nor comprehensive.

As a result, overlooked dependencies on third-party packages or failure with typical user input were

[3] In the early '90s when most of this code was initially written, this was an absolute must as TeX's main memory, register space, pool size, etc., was much smaller than today. These days one would produce quite different-looking code that would support extensions much better, by offering the necessary hooks, and this is what we are gradually introducing in various parts of the code. However, given that there are many packages out there that expect the code to look exactly like it does right now, changing anything means that these packages need to have corresponding changes.

[4] In fact the LaTeX Project team tried to update the package when it became clear that nobody was maintaining it, and we managed to produce a version that didn't die right out from the beginning. However, the package altered so many commands and used them in new ways that this emergency fix was only partially successful. So in the end we could only suggest that people should not use it in the future, or more exactly not until a new maintainer stepped forward and spends the necessary time to unravel the coding issues.

seldom found beforehand but only when the release moved to the distributions and everybody became (unwillingly) a tester — banana software after all, despite our best efforts above.

The LaTeX dev formats

To improve this situation and hopefully get to a release workflow that doesn't normally involve step 5, we developed the concept of a LaTeX development format. This format contains a prerelease of the upcoming main LaTeX release and is ready for testing by anybody using either TeX Live or MikTeX.

All the user needs to do is to replace his or her standard engine call by adding the suffix `-dev` to the name, for example, using

```
pdflatex-dev myfile
```

instead of `pdflatex myfile` on the command line. If you use an editing environment with integrated TeX processing, then there is normally some configuration possibility, where you can either make the same change or even add another menu item. Besides pdfTeX, all other major engines are supported as well, e.g., with `lualatex-dev` you get the new format on top of the LuaTeX engine, etc.

Pre-release identification

If you call LaTeX in this way you can immediately see that the pre-release format is used. For example processing this document with the line above gives:

```
This is pdfTeX, Version 3.14... (TeX Live 2019)
(preloaded format=pdflatex-dev)
 restricted \write18 enabled.
entering extended mode
(./TUB-latex-dev.tex
LaTeX2e <2019-10-01> pre-release-2
```

As you can see the format announces itself as a pre-release of the upcoming 2019-10-01 release of LaTeX, and the number tagged at the end indicates that it is the second pre-release we have distributed (the first was a trial to see if the mechanism functions correctly). If bugs are found during the testing (or if we enable further features for the upcoming release) we might issue another pre-release in which case the number would increase accordingly.

However, the important point to note here is that the development format is not like a "nightly build" (that you would get by tracking the LaTeX source at GitHub); rather, it changes only if we think that the code is ready for public testing, i.e., has passed our own internal tests in steps 2.a and 2.b.

How does it work?

The files for the pre-release are uploaded by the LaTeX Project Team to CTAN under the package names `latex-base-dev`, `latex-graphics-dev`, and if necessary `latex-tools-dev`, etc. From there they are integrated into the distributions into the tree `tex/latex-dev/...`, which is not searched by default. Thus, when you are using, say, `pdflatex`, only the files from the main release are used.

However, if any of the programs ending in `-dev` are called, then this extra tree is prepended to the search tree, so that not only the pre-release format is used, but also any other file from that tree, e.g., `article.cls`, is found first. For any package not part of the pre-release, the TeX engine will continue to find it in the main tree and use that version.

This allows any user who works on an important project (such as a thesis or a book) to quickly test if this work continues to typeset correctly under the upcoming format. Similarly, it enables any developer of a package that has known or unknown dependencies on a certain kernel version to check if any adjustments made work well with both the current and upcoming LaTeX release — and if so, upload a new version of his or her work prior to the actual release date of the new LaTeX kernel.

Reporting issues in the dev format

If, during such testing, issues or incompatibilities are found (that in the past would have led to step 5) we suggest that a Github issue is opened for them so that they can be tracked and addressed by the team. Details on how to open such an issue can be found at the LaTeX Project website [5].

References

[1] Frank Mittelbach. A regression test suite for LaTeX 2ε. *TUGboat*, 18(4):309–311, 1997. `tug.org/TUGboat/tb18-4/tb57mitt.pdf`

[2] Frank Mittelbach, Chris Rowley, and Michael Downes. Volunteer work for the LaTeX3 project. *TUGboat*, 13(4):510–515, 1992. `tug.org/TUGboat/tb13-4/tb37mitt-l3.pdf`

[3] Frank Mittelbach, Will Robertson, and LaTeX3 team. `l3build` – A modern Lua test suite for TeX programming. *TUGboat*, 35(3):287–293, 2014. `tug.org/TUGboat/tb35-3/tb111mitt-l3build.pdf`

[4] Joseph Wright. Automating LaTeX(3) testing. *TUGboat*, 36(3):234–236, 2015. `tug.org/TUGboat/tb36-3/tb114wright.pdf`

[5] LaTeX Project Team. Bugs in LaTeX software. `www.latex-project.org/bugs`

◇ Frank Mittelbach, LaTeX Project
Mainz, Germany
`frank.mittelbach (at) latex-project dot org`
`www.latex-project.org`