Automating LaTeX(3) testing

Joseph Wright and LaTeX3 team

1 Introduction

Testing has always been an important part of the work of the LaTeX team. Over the last couple of years, ideas first developed in the early 1990s have been used to create a flexible testing tool, l3build (Mittelbach, Robertson, and LaTeX3 team, 2014; LaTeX3 team, 2015). In an ideal world, every change would be checked by a full run of the entire test suite. In reality, that's not always the case: mistakes happen. What's therefore needed is to automate testing every time a change is made.

These concerns are not unique to the (La)TeX world, and thus it's no surprise that a variety of automated code testing approaches are already available. It can be done using a 'real' local machine, a private virtual machine or, increasingly commonly, a hosted solution. There are many companies now offering automated remote testing on virtual machines, and as this means a minimal amount of setup, it's this approach that the LaTeX team have been exploring.

Automating testing is part of a wider concept called 'continuous integration', often referred to as CI. As with many ideas in code development, this tends to attract a lot of acronyms and odd tool names: I'll try to keep those to a minimum here.

2 Setting up

For open source projects like LaTeX3, many of the providers of hosted services offer free accounts. LaTeX has chosen to use Travis-CI (`http://travis-ci.org`), partly as it is well-known, partly as it is easy to set up, and partly as it fits into other parts of our setup (see below). Of course, there are many other worthy choices.

The two key things any automated test system has to know: where to find the code that is changing and how to run the required tests. Travis-CI integrates with GitHub (`http://github.com`), one of the large number of websites offering hosting for version control using Git (Git team, 2015). The team have had the GitHub site for some time (`http://github.com/latex3/latex3`), so integrating with Travis-CI was easy.

Setting up the testing itself means telling Travis-CI about the type of code being tested. This is done using a plain text file, `.travis.yml`, which has to be in the main directory of the code repository. For common programming languages, such as C, Java or Ruby, Travis-CI knows the normal testing setups and likely only needs to be told the language involved. For TeX, that's not the case: we (I) need to give the system more information. Moreover, the virtual machine setup used by Travis-CI doesn't have a TeX system installed as standard. So there is a bit of work to do, but luckily in a well-documented and simple format.

Setting up the actual tests means pointing the system at l3build, which can be done in two lines:

```
script:
 - texlua build.lua check -H
```

This will run in the main directory of the code repository, and will run all of the current tests for LaTeX3. As the test is, ultimately, just pass/fail, I've told l3build to halt immediately if any test fails (`-H`).

To get a TeX system installed, I need to run a script, which will clearly depend on the nature of the virtual machine. For Travis-CI, that's currently Ubuntu 12.10LTS, so I use a `bash` script. For the test system, I need to make sure that script gets run before our tests; this is what Travis-CI calls the `install` step:

```
install:
 - source ./support/texlive.sh
```

The script itself needs to run an automated TeX (Live) installation. That has two parts: first downloading and installing a minimal system, then adding on extra packages that I need. (We'll see later why a small system is useful.) The script is relatively straightforward:

```
# Obtain TeX Live
wget http://mirror.ctan.org/systems/\
texlive/tlnet/install-tl-unx.tar.gz
tar -xzf install-tl-unx.tar.gz
cd install-tl-20*

# Install a minimal system
./install-tl \
  --profile=../support/texlive.profile

# Add the TL system to the PATH
PATH=/tmp/texlive/bin/x86_64-linux:$PATH
export PATH

cd ..

# Core requirements for the test system
tlmgr install babel babel-english \
  latex latex-bin latex-fonts \
  latexconfig xetex
tlmgr install --no-depends ptex uptex
```

The run of `install-tl` above uses a so-called profile file to tell the TeX Live installer what to do. That's again quite short:

```
# Profile for minimal TeX Live installation
selected_scheme scheme-minimal
TEXDIR /tmp/texlive
TEXMFCONFIG ~/.texlive2015/texmf-config
TEXMFHOME ~/texmf
TEXMFLOCAL /tmp/texlive/texmf-local
TEXMFSYSCONFIG /tmp/texlive/texmf-config
TEXMFSYSVAR /tmp/texlive/texmf-var
TEXMFVAR ~/.texlive2015/texmf-var
option_doc 0
option_src 0
```

Here, we do not install the sources or documentation (clearly not needed for runtime testing) and the installation location is non-standard: I don't have `sudo` on the virtual machine and the profile installation doesn't (yet) support `~` (the home folder) in the installation path.

You might wonder if I could have used `apt-get` to add the Ubuntu managed TEX Live. That runs, and means I wouldn't need a script (`.travis.yml` has an entry type for running `apt-get`). However, it installs TEX Live 2009, which is too old to run l3build. (There have been a lot of changes in LuaTEX since then.) For the team tests, we always assume an up-to-date and current TEX Live, so it makes sense to have the same on the Travis-CI setup.

You might also wonder how I worked out exactly what the minimal requirements were for the installation. That was a bit of work, but the idea was simple enough: run the tests on a local virtual machine and add packages one at a time until everything works!

3 Refining

Once the above was set up, Travis-CI started running automated tests each time changes were made to the GitHub version of the LATEX3 code. There were of course a few teething issues: it turned out that l3build was returning error level 0 ('success') even when tests failed! That was soon fixed in our code: a first demonstration of the use of automated testing.

With a virtual machine, each time tests are run the machine is 'reset' to a known state. That meant that each code change needed to do a fresh install of TEX Live: one of the reasons for keeping the system small. Ideally, I wanted to avoid the load on CTAN if possible. To do that, we've added *caching* to our `.travis.yml` file:

```
cache:
  directories:
    - /tmp/texlive
    - $HOME/.texlive2015
```

This compresses the directories listed at the end of each test run, then adds them to the 'clean' machine at the start of the next run.

Caching gives us a way to make sure a TEX system is available, but what about when the cache has to be reset, when new packages are needed or when updates are available? A bit of `bash` scripting sorts all of that. First, the basic installation can be wrapped up in a test looking for a TEX system:

```
# See if there is a cached version of TL
PATH=/tmp/texlive/bin/x86_64-linux:$PATH
export PATH
if ! command -v texlua >/dev/null; then
  # Earlier script code
fi
```

We can then run the update process:

```
# Keep no backups (makes cache smaller)
tlmgr option autobackup 0
# Update the TL install
tlmgr update --self --all \
  --no-auto-install
```

and finally add extra packages

```
tlmgr install \
  adobemapping  \
  amsmath       \
  ...
```

With this approach, the list of extra packages can keep growing, and any new entries will get added automatically. If the cache has to be cleared, an entirely new TEX Live and all of the required packages will be installed.

With the standard settings, Travis-CI emails the person who made code changes that led to the tests failing. For the LATEX3 source repository, we have a mailing list for every commit, so it makes sense to send those failure messages to the list too, which is done like this:

```
notifications:
  email:
    recipients:
      - latex3-commits@tug.org
    on_success: change
    on_failure: always
    on_start:   never
```

4 In use

Setting up all of the above took only a few days, and much of that was working out how best to install a TEX Live setup automatically and then to cache it. The actual `.travis.yml` configuration took only minutes to do.

Running the full test suite for LATEX3 currently takes around 6 minutes on the virtual machine, depending on whether the TEX Live system needs to be

```
 2
 3 Build system information
65
66 $ git clone --depth=50 --branch=master https://github.com/l
76
77 This job is running on container-based infrastructure, whic
78 If you require sudo, add 'sudo: required' to your .travis.y
79 See http://docs.travis-ci.com/user/workers/container-based-
80 Setting up build cache
89 $ rvm use default
91 $ ruby --version
92 ruby 1.9.3p551 (2014-11-13 revision 48407) [x86_64-linux]
93 $ rvm --version
94 rvm 1.26.10 (latest-minor) by Wayne E. Seguin <wayneeseguir
95 $ bundle --version
96 Bundler version 1.7.6
97 $ gem --version
98 2.4.5
99 $ source ./support/texlive.sh
161 $ texlua build.lua check -H
162 This is TeX, Version 3.14159265 (TeX Live 2015) (preloaded
163 (./l3build.ins (../local/docstrip.tex
164 Utility: `docstrip' 2.5d <2005/07/29>
165 English documentation    <1999/03/31>
166
167 ************************************************************
168 * This program converts documented macro-files into fast *
169 * loadable files by stripping off (nearly) all comments! *
170 ************************************************************
171
172 ********************************************************
173 * No Configuration file found, using default settings. *
174 ********************************************************
175
176 )
177
178 Generating file(s) regression-test.tex
179
180 Processing file l3build.dtx (package) -> regression-test.te
181 Lines  processed: 1363
182 Comments removed: 986
183 Comments  passed: 50
184 Codelines passed: 323
185
186  )
187 No pages of output.
188 Transcript written on l3build.log.
189 This is TeX, Version 3.14159265 (TeX Live 2015) (preloaded
190 (./01-expect.ins (../local/docstrip.tex
191 Utility: `docstrip' 2.5d <2005/07/29>
```

Figure 1: Log output from Travis CI (abridged).

re-installed. That's about the same time as it takes on a MacBook Pro i7 (my own laptop). So checking the code in almost real-time is certainly workable.

Most of the time the tests pass, and the web page shows a simple report to this effect. When a test fails, as well as the email sent, the web page shows the failure. Notice that in both cases we get the commit reference, which we can use to go straight to the code changes on GitHub. There is also an overview of changes over time, so you can quickly get a feel for what broke and fix the system.

Sometimes of course you need more detail, and for that the terminal log is visible (Figure 1). As you can see, each phase of the process is separated out so you can collapse parts that are not important: for example, if the `install` phase is fine but there is a problem with the tests. The log loads in real time when a test is running, so you can monitor what's happening and if necessary kill a test, for example if something seems to have hung.

That flexibility and speed means it's been possible to add more tests, checking on how the core LaTeX code interacts with some contributed packages.

5 Conclusions

Setting up and running an automated test system using a hosted virtual machine makes running tests on every change easy. It shouldn't be seen as an alternative to running tests *before* changing code, but it is another tool to exploit in keeping ahead of the inevitable bugs.

References

Git team. `http://git-scm.com`, 2015.

LaTeX3 team. "The l3build package". Available on CTAN: `http://ctan.org/pkg/l3build`, 2015.

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

◇ Joseph Wright
Morning Star
2, Dowthorpe End
Earls Barton
Northampton NN6 0NH
United Kingdom
`joseph dot wright (at) morningstar2.co.uk`

◇ LaTeX3 team
`www.latex-project.org`