Fly me to the moon:
(La)TeX testing (and more) using Lua

Joseph Wright

\LaTeX\ Project
Testing \LaTeX

Requirements

– Test both programming and typesetting
– Standard TeX system (no special binaries)
– Ability to deal with kernel-specific requirements
– Minimal dependencies
Testing \LaTeX

Requirements

- Test both programming and typesetting
- Standard \TeX system (no special binaries)
- Ability to deal with kernel-specific requirements
- Minimal dependencies

Approaches

- Write data to log
- Include markers for post-processing
- Normalise . . .
- Compare with reference version
\input{regression-test}
\RequirePackage{siunitx}
\START
\ExplSyntaxOn
\OMIT
\cs_set_protected:Npn \test:n #1
\OMIT
\cs_set_protected:Npn \test:n #1
{\siunitx_number_format:nN {#1} \l_tmpa_tl \tl_show:N \l_tmpa_tl }
\TIMO
\TEST { Basic-formatting:-integers }
\OMIT
{\test:n { 1 } \test:n { 123 } \test:n { 123456789 } \test:n { 12345678901234567890 } \test:n { 00001 } \test:n { 00000 } }
\OMIT
\TIMO
\input{regression-test}
\RequirePackage{siunitx}
\START
\ExplSyntaxOn
\OMIT
\cs_set_protected:Npn \test:n #1
\OMIT
\cs_set_protected:Npn \test:n #1
{\siunitx_number_format:nN {#1} \l_tmpa_tl \tl_show:N \l_tmpa_tl }
\TIMO
\TEST { Basic-formatting:-integers }
\OMIT
{\test:n { 1 } \test:n { 123 } \test:n { 123456789 } \test:n { 12345678901234567890 } \test:n { 00001 } \test:n { 00000 } }
\OMIT
\TIMO
TEST 1: Basic formatting: integers

\l_{tmpa_tl}=1.
\textit{<recently read> \}}
1. ... \}}
\l_{tmpa_tl}=123.
\textit{<recently read> \}}
1. ... \}}
\l_{tmpa_tl}=123,456,789.
\textit{<recently read> \}}
1. ... \}}
\l_{tmpa_tl}=12,345,678,901,234,567,890.
\textit{<recently read> \}}
1. ... \}}
\l_{tmpa_tl}=1.
\textit{<recently read> \}}
1. ... \}}
\l_{tmpa_tl}=0.
\textit{<recently read> \}}
1. ... \}}
History

- Core macros go back to early 1990s
- First sed . . .
History

- Core macros go back to early 1990s
- First sed . . .
- Then Perl and make . . .
History

- Core macros go back to early 1990s
- First sed ...
- Then Perl and make ...
- Then Perl and batch scripts ...
History

- Core macros go back to early 1990s
- First `sed` ...
- Then Perl and `make` ...
- Then Perl and batch scripts ...
- Plus `cons` ...
More requirements

– Just use a \TeX\ system
– Work for multiple engines
– Cross platform
– A \textit{product} not an internal tool
– Support wider release work-flow
usage: l3build <command> [<options>] [<names>]

The most commonly used l3build commands are:
- check  Run all automated tests
- clean  Clean out directory tree
- doc    Typesets all documentation files
- install Installs files into the local texmf tree
- save   Saves test validation log
- tag    Update release tags in files
- uninstall Uninstalls files from the local texmf tree
- unpack Unpacks the source files into the build tree

Valid options are:
- --config|-c Sets the config(s) used for running tests
- --date Sets the date to insert into sources
- --dry-run Dry run for install
- --engine|-e Sets the engine(s) to use for running test
- --epoch Sets the epoch for tests and typesetting
- --first Name of first test to run
- --force|-f Force tests to run if engine is not set up
- --halt-on-error|-H Stops running tests after the first failure
- --last Name of last test to run
- --pdf|-p Check/save PDF files
- --quiet|-q Suppresses TeX output when unpacking
- --rerun Skip setup: simply rerun tests
- --shuffle Shuffle order of tests
- --texmfhome Location of user texmf tree

See l3build.pdf for further details.
#!/usr/bin/env texlua

-- Build script for "siunitx" files

-- Identify the bundle and module
bundle = ""
module = "siunitx"

-- Install config files
installfiles = {"*.cfg", "*.sty"}

-- Release a TDS-style zip
packtdszip = true

-- Typeset only the .tex files
typesetfiles = {"*.tex"}

-- Detail how to set the version automatically
function update_tag(file, content, tagname, tagdate)
    return string.gsub(content,
        "\n\ProvidesExplPackage %{siunitx%} %{\%d\%d\%d\%-%d\%d\%-%d\%d\%} %{\%d\%.\%d\w?%}\n",
        "\n\ProvidesExplPackage {siunitx} {
        .. tagdate .. "} {" .. string.gsub(tagname, "-v", ") .. "}\n"
    end

function tag_hook(tagname)
    os.execute('git commit -a -m "Step tag"')
    -- os.execute('git tag -a -m "" ' .. tagname)
end

-- Find and run the build system
kpse.set_program_name ("kpsewhich")
if not release_date then
dofile(kpse.lookup("l3build.lua"))
end
Current status

Working well

– Core testing
– Multiple engines
– Multiple configurations
– Building PDFs and zips for CTAN
– Basic file tagging

Still to do

– More flexible test selection
– Re-vamp PDF-based testing
– Tagging files on installation
– Working with ‘dynamically tagged’ files
Current status

Working well
- Core testing
- Multiple engines
- Multiple configurations
- Building PDFs and zips for CTAN
- Basic file tagging

Still to do
- More flexible test selection
- Re-vamp PDF-based testing
- Tagging files on installation
- Working with ‘dynamically tagged’ files
– Frank Mittelbach
– David Carlisle
– David Manura
– Will Robertson
– Many StackOverflow answers ...
– Travis-CI

https://github.com/latex3/l3build