Diagnosing TEX Errors with a Preprocessor

David Ness
803 Mill Creek Road, Gladwyne, PA 19035
215-649-3522.

Abstract

TEX finds our errors with ease; however, it sometimes reports them in ways that are hard to understand. Generally this is because we have confused it by unintentionally misrepresenting something. For example, if we do something as simple as forget an escape on a dollar sign TEX will probably give us some obscure diagnostics about math mode. This paper discusses some preprocessors that can warn us about potential problems before we submit our files to TEX. These programs may be of particular help to new users.

Purpose

TEX errors can be difficult to diagnose, particularly for the new user. TEX is *very* flexible and general. It provides a rich world for developing and describing typesetting processes. The very richness, however, of this world — and its flexibility and generality — seems to work against the new user, particularly by making it hard to see the cause and cure for problems. The *suite* of programs described in this paper attempts to deal with such problems.

The Overview

Instead of writing one (complex) program to help diagnose TEX errors, we wrote a number of different programs, each of which could deal with one, more restricted, problem domain. The results of these separate analyses can be integrated into a single picture. By separating parts of the error analysis process we allow for independent evolution, and people who have only one particular problem need use only the module appropriate to it.

The modules that comprise this *suite* have the following functions:

`TEXCHECK` checks for some common TEX errors, particularly those made by new users. For example, this module looks for unescaped dollar signs and percent signs (*i.e.*, `$`, `%` instead of `\$`, `\%`).

`TEXBRACE` takes the input file and keeps careful track of the use of left and right curly braces. If the left and right braces aren't properly balanced, the locations of those not matched are reported. The source file is also rewritten (temporarily) in an attempt to make brace problems easy to spot.

`TEXLOG` analyzes the log that results from running TEX. This program, written in `AWK-WEB`, is still in a preliminary state and is not described further.

`TEXERROR` merges the results of running `TEXCHECK`, `TEXBRACE` and (when it is ready) `TEXLOG`, along with the original source file, into a new source file that can be edited to correct the mistakes and remove any error comments.

`TEXFIND` is an experimental program designed to help users relate their input source to the output obtained from a TEX run. This aid is quite distinct from the others discussed here.

Organizing a TEX Source File

Users, particularly new users, can find it helpful to adopt some discipline in organizing their source files. This will prove useful when diagnosing and fixing errors. As one gains comfort and familiarity with TEX, this discipline can be relaxed, but at least in the early months it is wise to adhere to some simple principles.

Keep macro definitions in a separate file, to be incorporated by an `\input` command; this makes life much simpler. Modifications to macros as they are debugged represent an effort quite different from that required to modify the basic text. If these problems are isolated, it is easier to see what's going on.

Some of the diagnostic help provided by the programs in our *suite* is rendered more effective when the source is split into logical pieces. For example, the `TEXCHECK` program flags all occurrences of unescaped number signs (`#`). Generally these don't occur in normal text, but they are a regular

part of macro definitions. If the macros are in a separate file, which we don't pass through TEXCHECK, then no confusing diagnostics will appear.

TEXCHECK World-view

TEXCHECK was written to warn about potential problems. Since it is preferable to be warned too often than not often enough, occasionally warnings are generated about things that would be found legal if a more substantial analysis of the TEX source were made. The source file is not analyzed in a deep way, so complex things like mode shifts and macros will be missed.

The idea of creating a *Lint* for TEX was rejected because of the complexity of TEX syntax. After all, in TEX it is an easy matter to redefine nearly everything, and keeping track of all of this would rival writing the TEX processor itself!

We gave up on the idea of doing the job perfectly and may have gone to the other extreme. Our principal goal is simplicity, in particular we hope to share these ideas with others, so that feedback will help us to develop them further.

TEXCHECK warnings. TEXCHECK warns about a variety of possible errors. Most of the warnings are designed for new users, but even old hands at TEX may find a pass through TEXCHECK is worth the trouble, particularly if the source file was captured by someone not too familiar with TEX.

Here is a list of warning messages which TEXCHECK may issue:

Angle brackets probably need to be in `\tt` *font.*

Many TEX fonts place the upside down exclamation point and question mark in the ASCII table where the angle brackets are in the `\tt` font. This warns about all angle brackets in the text, unless they are preceeded on their line by a percent sign (and thus are probably in a comment).

Something may be missing to avoid end-of-sentence spacing.

TEX has some sensible, but complex, rules about when it puts in end-of-sentence spacing. This warning indicates that TEX will put end-of-sentence spacing after a particular period, and TEXCHECK thinks it may be inappropriate (for example, on the period after 'Dr').

Perhaps there should be end-of-sentence spacing here, and there won't be.

We might also have a place where a capital ahead of a period blank might have suppressed end-of-sentence spacing when it shouldn't have. This checks for that situation too.

Em- and En-dashes generally abut the words on either side.

English typesetting specification suggests that dashes abut the words on either side. This warns about what appears to be contrary usage.

Number-signs are generally only in macro definitions.

Number signs are common in normal text so they may sometimes be entered without being properly escaped. Since they normally represent arguments to macros in TEX, diagnostics can be confusing. TEXCHECK warns about them indiscriminately, *i.e.*, it makes no attempt to see if the unescaped # is being used legally (for example in a macro definition).

Double quotes should go away.

TEX usage calls for two left quotes and two right quotes, which become left double quote and right double quote. While the typewriter double quote character will produce the TEX right double quote, it probably shouldn't be used at all for quotes in a TEX source.

Ampersands usually perform tab skips.

Ampersands generally represent tab stops in alignments. This warns about *all* unescaped ampersands because error diagnostics that result from ampersand misuse can be confusing.

Underscores and carets generally are sub- and superscripts in math mode.

Sometimes underscores and carets creep into normal TEX text. They can generate confusing error messages there because they ordinarily represent sub- and super-scripts in math mode. This message will appear when underscores and carets are detected not following a dollar-sign that might indicate a previous shift into math on the line. In TEXCHECK no attempt is made to detect whether we are in math mode, which is complicated to determine.

'%' preceded by digits probably should be escaped.

We often forget to escape percent signs. This can cause text to disappear. This warning raises a question about situations where a number is followed by a percent sign (perhaps separated by blanks), without the percent sign being escaped.

Check to make sure that the thing following the '%' sign is a comment.

As an alternative, if the first thing following a percent sign (after some optional blanks) isn't an upper case alphabetic character (that might begin a comment), then we also raise a question.

The dollar sign indicates a shift to 'math'. Was that intended?

If a dollar sign happens to be followed by a number, it is possible that a real dollar amount was intended, and an escape forgotten. This message will, of course, improperly appear when something introduced in math mode begins with a number, but this is a smaller price to pay.

Running TEXCHECK. TEXCHECK can be executed with a number of switches. If none are specified it will, like the other modules described here, prompt for the appropriate inputs.

The switch `-I x` tells TEXCHECK to use 'x' as the input file. `-O x` names 'x' as the output file. If the switch `-F x` is used, then the input file is assumed to be 'x.TEX' and the output file will be 'x.CHK'.

TEXCHECK can issue reports at five different levels. The level of reporting is indicated with a `-R n`. Level 3 provides the greatest amount of descriptive information while level 0 provides the least. Level −1 is used when the output of TEXCHECK is to be fed into TEXERROR.

TEXBRACE Functions

The purpose of TEXBRACE is to help us find errors in curly-brace structure. These pose particularly nattering problems for TEX because a missing brace will often cause TEX to misinterpret some element of the structure and can create obscure error messages.

TEXBRACE performs two functions. The easier to understand involves finding the lines on which unmatched left curly braces occur. A list of line numbers for unmatched braces is made by the program and as corresponding right braces occur this list is adjusted. If the list is not empty at the end of the file, it shows where the unmatched braces were. TEXBRACE will also report on excessive right curly braces if they occur, but this is generally a less difficult problem.

The other function of TEXBRACE involves creating a copy of the input file in a form that will emphasize its brace structure. When writing this copy, TEXBRACE replaces returns with blanks, thus producing (impossibly) long lines of text; however, each time a brace is encountered we drop to a new line and indent (for left braces) or outdent (for right braces). The file that results from this isn't good for anything but looking at brace structure, but any problems with this structure then turn out to be obvious.

Running TEXBRACE. TEXBRACE can be executed with a number of switches. The switch `-I x` tells TEXBRACE to use 'x' as the input file. `-O x` names 'x' as the output file. If the switch `-F x` is used, then the input file is assumed to be 'x.TEX' and the output file will be 'x.BRC'.

TEXBRACE can issue reports at two different levels. The level of reporting is indicated with a `-R n`. Level −1 is used when the output of TEXBRACE is to be fed into TEXERROR. Level 0 provides output to be read by the user. At level 0 the entire text of the file is rewritten in a way that emphasizes brace structure. At level −1, only the error messages are written.

TEXERROR Functions

TEXERROR takes the output of TEXCHECK, TEXBRACE and (when ready) TEXLOG and merges them with the original source into a new copy of the file. Each of the routines identifies the line number on which potential errors have been reported and this module takes all messages appropriate for each line and places them in the output file just following the line in question.

The lines generated by these programs are in a format appropriate for TEX comments. TEXERROR also arranges to have the first line of a block of error messages begin with "%ERROR" and end with "%ERROR-MERGE End". This makes them easy to find with a text editor.

Running TEXERROR. TEXERROR can be executed with a number of switches. The switch `-I x` tells TEXERROR to use 'x' as the input file. `-O x` names 'x' as the output file. If the switch `-F x` is used, then the input file is assumed to be 'x.TEX', the brace error input file is 'x.BRC', the check error file is 'x.CHK' and the log error input file is 'x.ERL'. The output file will be 'x.NEW'.

TEXFIND

TEXFIND is an experimental program designed to act as a prototype for a TEX error facility that would allow the user to associate the input source file directly with what is seen in the output.

TEXFIND takes each piece of recognizable text in a document and follows it with a TEX call `\spc[m,n]` where `m` represents the line number and `n` the column number of the source file line that began the word in question. Since it is very difficult to know anything about the actual effect of a TEX macro without profound analysis, only first level text is recognized by TEXTFIND.

Once the \spc[...] markers have been placed in a source file, that file can be run through TEX with a definition of \spc[...] that will cause TEX \special commands to be written into the DVI file. This information is then available to display drivers able to use it to relate things being displayed back to their original locations in the source file.

An important modification needs to be made to a source file before it is sensible to run TEXFIND on it. There are situations (in the middle of a macro definition, for example) where inserting the \spc[...] markers might prove disastrous. For that reason a special form of TEX comment, %!, is used to toggle \spc generation on and off. TEXFIND begins operation with \spc generation off, so the source file should be modified by putting a '%!' on the line prior to the one on which the text begins. At the moment there are no drivers which will allow us to see the markers, so this represents an experiment in its earliest stages.

The Implementations

The programs described here are implemented in C-WEBs. The language we used is Norman Ramsey's implementation *via* SPIDER. The copyright on these WEBs has been assigned to TUG, the TEX Users Group, so that they may be freely exchanged in a community as wide as possible. We hope that feedback from this community will result in improvement of these programs.

Relationship to Text Editors

The output of TEXERROR can be processed by any ASCII oriented text editor. A good editor may deal effectively with the kind of messages that TEXERROR produces.

For example, using the old standby PE2, a definition like:

```
d a-e = [1/\%ERROR/] [mark line]
        [1/ERROR-MERGE End/] [mark line]
```

makes it possible to locate the next block of error messages and highlight them simply by typing <ctrl>-E. The normal editor function <ctrl>-D will then delete this block of error messages. Thus it is possible to page through the file with successive <ctrl>-Es and <ctrl>-Ds.

The TE.BAT File

The programs in this *suite* work together conveniently. One easy way is to construct a DOS .BAT file that calls them in sequence. The following simple file TE.BAT does this:

```
@ECHO OFF
REM "<$TeX Error Analyzer - Ver (1)$>"
TEXBRACE -f %1 -r -1
TEXCHECK -f %1 -r -1
Echo TEXLOG doesn't exist yet
TEXERROR -f %1
DEL %1.BRC
DEL %1.ERL
DEL %1.CHK
```

Here the programs in the *suite* are executed in sequence, and the results are fed into the TEXERROR run where they are merged. Execution of TE filename results in a file filename.NEW which should be copied over the original filename.TEX after corrections have been made and it is decided that the new file is better than the original.

Experiences

A first, and rather pleasant, surprise was that these programs, particularly TEXCHECK and TEXBRACE, proved to be helpful to long time users of TEX, as well as to novices, since both groups still make mistakes in files which these programs isolate quickly and without much fuss.

The programs have also proved useful by allowing sophisticated TEX users to have texts typed by typists not very experienced with TEX. The rules about typing \%, \$, \& instead of %, $, & tend to be forgotten until these things have been typed many times. With these programs it doesn't seem to matter whether they are remembered, since it's so easy to find these mistakes and fix them.

Acknowledgements

The ideas presented here resulted from discussions involving S. Bart Childs of Texas A&M University, Alan Hoenig of John Jay College, and the author. Suggestions from many others with whom we have discussed this idea over the past six months are gratefully acknowledged.