Debugging LaTeX files — Illegitimi non carborundum

Barbara Beeton

Abstract

Every LaTeX user has, at least once in her career, been faced with a thorny problem when compilation shuts down for some obscure reason. How to deal with simple problems is reasonably well known, but there are situations when the time-honored methods fall short.

This article will present strategies and tactics for dealing with the many types of problems that have arisen during long experience as a member of the AMS technical support staff, handling questions from authors and the editorial staff. Both common and uncommon glitches will be visited, with a bias toward avoiding problems in one's own work — something for everyone.

1 Background

Last year, the AMS published on the order of 60,000 pages of books and journals, most of them produced from LaTeX files prepared and submitted by authors. The acceptance of a journal article is based on scientific merit, judged by an editorial committee and referees reviewing a paper or electronic document; it might even be handwritten. No consideration is supposed to be given to the presentation, only to the content. Books are contracted by the acquisitions staff, all of whom are professional mathematicians familiar with LaTeX, but by no means TeXnically skilled. What comes in for production is what we have to deal with.

Assume that the accepted work *is* prepared in LaTeX (if it is not, it will be (re)keyboarded by a competent entry operator and delivered in usable condition); the quality of submissions varies greatly, providing a wealth of opportunity to test (and improve) one's debugging skills.

Production is carried out on networked Linux systems. The available macro library is in three parts: TeX Live, which is updated at most once a year; local versions of "public" macro files and fonts (sometimes including updated versions that will become part of next year's TeX Live collection); and macros, fonts and other tools that are entirely local to AMS. Everything is archived with Subversion, with archives of published books and articles extending back a couple of decades. The versions of (La)TeX and all used packages are recorded within the main file for a published work using the snapshot package, so that if reprocessing is necessary, the original environment can be recreated. This setup provides the stability necessary to produce a steady flow of new publications while handling reprints, revised editions, and conversion of existing publications to other formats such as ebooks.

As described so far, this workflow is effective and reliable once the files representing a manuscript are ready to be sent to the printer. But all sorts of things can go wrong before that happy moment. One guiding principle tops all others: If something goes wrong, it must be possible to recover a known, stable starting point quickly and reliably.

2 Preparation — plan ahead

There are certain conventions that, if followed diligently, can make one's life easier in the long run. First, choose good tools and become familiar with them.

The most important tool is a competent editor or IDE. The author uses emacs, but other options are available, some for single users on one platform, some intended for cooperative authoring online, and a number of alternatives somewhere in between. A list of such tools can be found in answer to a question on the `TeX.stackexchange` site (hereafter "`tex.sx`" [4]).[1]

The author also prefers to process files from the command line. This makes it possible to correct simple errors, such as misspelled control sequences, interactively, avoiding delays and the possibility of a cascade of irrelevant error messages as a consequence of a possibly trivial error. (But don't forget to correct the file as well before the next run.)

Among the features most useful for debugging are these:

- good search facilities;
- brace and `\begin`/`\end` matching;
- multiple windows viewable at the same time;
- "go to" a specified line number;
- ability to match strings and to ask "how many?"

Another important consideration is how directories and files are laid out and addressed. It's advisable to *avoid spaces in file names*; not all operating systems handle such spaces gracefully (or at all). Similarly, some operating systems are case sensitive — to avoid problems here, *use only the lowercase alphabet for file names*; digits and hyphens are also "neutral" in this regard, but (extra) periods and characters with special meanings to TeX (e.g., the underscore) are best avoided.

Keeping files at a manageable size will pay off in the long run. For a large work like a book or dissertation, place each chapter in a separate file,

[1] LaTeX Editors/IDEs, `http://tex.stackexchange.com/q/339`

controlled by a main or "driver" file. This will permit you to work on just one chapter at a time, taking advantage of the `\includeonly` facility. If you have large tables or figures, placing each in a separate file can also be handy, as it is then possible to exclude one with a single `%` to comment it out (this also makes it easy to move it to a different place in the main file if that becomes necessary).

Finally, when preparing files, it's usually a good idea to end files other than the main file with a separate line `\endinput`; this avoids problems from garbage that is sometimes added on, unasked, when a file is shipped from one system to another. And *never* put a line `\end{document}` in any file but the main driver file.

Another suggestion: Learn where the log file can be found, before you ever need to look at one. Some IDEs hide this from a user; if your job goes south and you cannot check what is happening by looking in the log, you are going to have a very difficult time figuring out how to make things right.

And one more:

Don't update your system in the middle of an important project.

New versions of packages can have new, incompatible features, and old packages can disappear. Of course, if your hardware decides to conk out at that point, this is not useful advice. But you *do* keep a full current backup, don't you?

3 Isolate and insulate your testing

Use *copies* of your files to test.

If the error you're trying to fix isn't something like a simple typo, protect yourself against possible disasters: set up a special debugging environment. At the very least, make a backup of your files, maybe even a zip of the full working directory tree, and put it in a safe place. You know your current situation, and you want to be able to return to it safely.

Under no circumstances make experimental changes to your only copy of any files.

Better yet, if you have the space, create a separate test area, identical in all important respects to the "live" work area, and do your experimenting there.

If the job consists of more than one file, start by copying *only* the driver file — the file that reads in all the others — into the test area. This will be your guinea pig.

Use a "soft link" to access other files in the job. For a Linux system, this involves issuing the command

`ln -s` ⟨*directory name*⟩

and adding the name of that location to the path. (It should be possible with a web search to find out how to do this for other systems.)

Process the job interactively. Then simple errors can be corrected at once, before they spawn meaningless and confusing error messages. (Remember to make the corrections in *both* test and real files.) And if an error is detected that can't be corrected interactively (such as an unrecognized or unended environment), the job can be stopped at once and the problem fixed before continuing.

Processing a job in nonstop mode (the usual procedure when launching a compile from within an IDE) will, of course, list all errors in the log file (up to a maximum of 100), but a single error that is not the simple misspelling of a symbol name can cause a cascade of spurious messages that would not have been necessary unless the first error was encountered.

More about this approach below, under "Divide and conquer".

4 Some tools for interactive diagnosis

Some diagnostic commands are available to send information to both the terminal and the log file.

- `\message{...}` writes out a message in the log and on the screen; it can be used to report when processing has reached a predetermined point. For example,

```
\message{last section, page \number\thepage^^J}
        last section, page 904
```

- `\show` reports the current meaning of a command; processing is suspended to permit additional interaction. Example:

```
\show\LaTeX
  > \LaTeX=macro:
  ->\protect \LaTeX .
\show\protect
  > \protect=\relax.
```

  Following the halt, more input can be inserted by typing `i` followed by a command or text. "Enter" will restart the session.

- `\showthe` reports the *value* of a command; processing is likewise suspended.

```
\showthe\hfuzz
  > 1.0pt.
```

A number of tracing commands are available to provide details of the processing flow. (Caution: tracing requests can deliver more information than you usually want, so be selective.) The result is sent only to the log unless requested otherwise. These are the tracing commands used most often by the author:

- `\tracingoutput` can be set to 1 to report, in symbolic form, the contents of all boxes that are written to the output;

- `\tracingcommands` and `\tracingmacros` give the gory details of LaTeX processing;
- `\errorcontextlines=200` sets the maximum number of lines associated with a single error message; the default value (5) often shows too few lines to understand the entire operation;
- `\tracingonline` directs the report of the other tracing commands to the screen as well as to the log.

Details of these commands (and many `\tracing...` relatives) can be found in *The TeXbook* [2] or in *TeX by Topic*[2] [1].

5 The log file is your friend

The (La)TeX log file records every action taken — what files and fonts are read, assignment of boxes and counters, redefinition of important commands, and so on. More importantly, from a debugging point of view, errors are reported in (sometimes excruciating) detail, all identified by line number in the source file.

Always check the log file for error messages:

```
! Undefined control sequence.
l.457 \fobx
              {%
```

Warnings are noted too, but without line number:

```
LaTeX Warning: There were undefined references.
```

Interpreting these messages can be a challenge, but this information should direct your first line of inquiry. If the system you are using hides the log file, ask how to find it. And don't delete the log file without looking at it.

Not every line number reported in an error message clearly identifies the exact line where the problem is located. The scope of math content (what is between `$` signs or other math specifiers) is not permitted to include paragraph breaks, so a missing `$` may not be reported until the next paragraph break, which may be a number of lines later in the input. (This restriction is also the reason that blank lines are not permitted within multi-line math display environments.) The other error associated with math mismatches is

```
! Missing $ inserted.
```

when a closing `$` is forgotten. This too is limited to the current paragraph, and should be easy to diagnose and repair.

An error within a figure, table, or multi-line display will also usually report the line number at the end of the environment, rather than on the line where it occurs, but again, the scope is relatively limited.

[2] In TeX Live; access with `texdoc texbytopic`.

Another reason for a report far away from where the error occurred is an unmatched group — an errant `{`, `\bgroup`, `\begingroup` or `\begin{`⟨*env*⟩`}`. In the case of a mismatched environment, this error will be reported as

```
! LaTeX Error: \begin{⟨env1⟩} on input line
                 nnn ended by \end{⟨env2⟩}
```

This will be reported as soon as the (incorrect) end is encountered and the line number should be correct. A mismatched grouping element, on the other hand, will not be reported until the end of the job, and then not even as the usual warning. The report consists of several lines:

```
(\end occurred inside a group at level m)
### semi simple group (level m) entered
    at line nnn (x)
### bottom level
```

Here, m will identify how many of these open groups remained at the end of the job. x will identify the unmatched grouping element: `\begingroup`, or `{` for either `{` or `\bgroup`. Again, the line number should be correct, just not in the place where the omission occurred.

Other possible error messages are shown in the documentation of various packages. Most messages include some line number, and in general localization is reasonably good; this is often enough to locate an error so it can be corrected without having to progress to more complicated searching steps. As soon as the error is identified and the fix verified, you can correct your *real* file, continue with the main task, and forget about the copies, which have now performed their intended function.

But, you may ask, when the job consists of multiple files, how can one be sure in which file the reported line number actually exists? See the next section.

The important lesson here is this:

Don't delete the log file until after you've extracted every bit of useful information.

It has also been suggested to the author that saving a log file for even longer (under a different name) has merit, as it makes possible the comparison of two logs when there is a question about what changed between two runs.

6 E pluribus unum — but which one?

Let us assume that the error was reported in a text file, not a package.

When the log file reports a line number, the first reaction is to look in the main file. But if that file is only 95 lines long, and the reported line number is 2345, that does not compute.

Make a copy of the log file, and work backward from the relevant error message. If some pages have actually been output, the page number (shown in square brackets: `[17]`) can point to a chapter, which ideally should be in a file of its own. Failing that, eliminate material that is, for this purpose, useless.

Messages about overfull boxes can be ignored — delete those lines. A matching pair of parentheses will usually enclose a file name and some more material. Look for a "completed" parenthesized group, such as

```
(C:/tech-support/debug/preface.tex
Preface
[1] [2]
)
```

and delete the whole group. What will finally remain is an opening parenthesis followed by a file name — the name of the file that was open when the error was reported. The reported line number should be found there.

But what if the line number was reported only at the end of the job, a `level` m situation? Here's where an extra `\end{document}` comes into play.

Keep working only with test files. Don't touch the real files until the source of the problem is identified.

Start from the end of the driver file and insert `\end{document}` between two `\include` statements. The "binary" approach is appropriate here — start in the middle. (More about this under "Divide and conquer".) Process what's left. If the `level` m condition is still reported, the target file is in the first half; if it's absent, look in the last half. Comment out `\include` statements that have been absolved of blame, and move `\end{document}` around until the target file is identified. This gets more complicated, of course, if $m > 1$, but the principle is the same.

7 Housecleaning

At some point, you will find the file where you think the error should be. Maybe you have a tightly defined line number. But maybe you still have only a general idea of where to look. Since you want to process only one file, clean out the clutter so it won't cause confusion.

Modify the driver file. adding an `\includeonly` line that specifies only the suspect file. Comment out commands that will include irrelevant pieces that aren't launched with an `\include` command:

- unnecessary (for the test) packages;
- `\tableofcontents`;
- anything related to the bibliography;
- `\printindex`.

Clean out your suspect file too. Don't worry about destroying the file; this is a copy, right? Here are the things that can be removed — carefully.

- lines commented with `%` at the beginning;
- lines between `\begin` and `\end{comment}`, inclusive;
- lines between `\iffalse ... \fi`, inclusive (this is equivalent to a comment).

Make sure that all groups are completely specified. This means matching all `\begin` and `\end` environments and all methods of "bracing". Check for these elements using your editor's "`how-many`" function:

- number of opening braces `{` = number of closing braces `}` (sometimes the string `% }` is added when an opening brace stands alone in the code, so be aware of this possibility);
- number of `\begin{` = number of `\end{`;
- number of `\begingroup` = number of `\endgroup`;
- number of `\bgroup` = number of `\egroup`;
- number of `\[` = number of `\]`;
- number of `$` signs is even, as is number of `$$`.

Process what's left, and look at the log for help as you go along.

Many problems are the result of an "unmatched" condition, so you might get lucky and not have to go any further. But let's assume it's still unidentified.

8 Divide and conquer

What you want to do is isolate the paragraph, or the smallest portion of the file, that is triggering the error. (Work with a copy, and keep another copy, just in case.)

Find a good paragraph break halfway through the file. Insert `\endinput` preceded by a blank line. Make sure it doesn't split up a `\begin`/`\end` pair or any group. Process this reduced file. If no error is reported, the problem is in the last (unprocessed) half. Remove the part that works, and keep moving `\endinput` until the source of the problem is located. If the solution is obvious, fix it and test. Apply the fix to the full *test* copy and try processing it. Once you are sure the fix is correct, insert it in the *real* version and test again.

But what if the solution isn't obvious?

If what remains is still too large for you to identify the problem quickly — perhaps it is a long proof, with steps presented as a list — make a copy of the file under another name and keep only the test material in the "working" file. (Many times this author has modified her *copy*, which is not the one that the driver file will input. This leads to exasperation.)

Reduce the size of this file by commenting out items that look harmless. Don't delete anything yet — what you think is harmless may actually be part of the problem. Keep iterating this process until there is no way to get rid of more material without also eliminating the (not yet located) error. This is now your "minimum (non-)working example", an "MWE".

Examine what's left in the file, and

Pay attention to the clues in the log.

Of course, once you know what needs to be fixed and how to fix it, you can verify this by making the necessary changes in your test file and rerunning LaTeX to confirm. If this works, install the fix in your real file, process it, and *if you find no other problems*, you're on your way!

If you do find more problems, it's back to the start, but now you know how to proceed.

One area hasn't yet been addressed — an error reported before

`\begin{document}`

is found. See below.

And there are more techniques that you can apply yourself, before calling for help.

9 Sometimes, more drastic action is required

In this section, we're still discussing problems in the body of the document.

Once a problem has been reduced to an MWE, it's time to take advantage of the available diagnostic tools to obtain more information. In addition to the commands shown on page 160 in the section on interactive diagnosis, these are also useful. (More detailed information on these commands can be found in *TeX by Topic* [1].)

- `\tracingmacros` reports the details of macro expansion, along with the values of the arguments.
- `\showboxdepth` specifies the number of box levels to display, usually set to `\maxdimen` for tracing.
- `\showboxbreadth` specifies the number of successive elements displayed on each level.

There are more, but these are generally the most useful.

If you are desperate, and a real masochist, you can specify `\tracingall`, but sorting through this information will tax both your patience and your sanity, and usually a "simpler" approach can be found. See the definition of `\tracingall` in the file `plain.tex` to see what is unleashed.

But if you have to resort to tracing, there may be an easier way.

10 In case more help is needed

Some useful resources are available online. You may not be the first to encounter a particular problem.

The archives at `tex.sx` [4] are a good place to look. If you don't find anything resembling the problem in your file, ask a new question. (You should register if you're not already a participant in the forum.) For best results, include a complete MWE; you already have one — the mimimum (non-)working example that you have been struggling with. Clean out any commented material, and, if appropriate (and possible), "anonymize" it by substituting dummy text; make it as minimal as possible while still demonstrating the problem. Include relevant lines from the log of the example you're posting, and an explanation of what you've tried. The participants in the forum are knowledgeable and friendly, and they enjoy a good puzzle — but they do need enough information to be able to experiment, and providing an MWE that they can copy and paste will yield results more quickly than if guesswork is needed.

11 Errors reported before `\begin{document}`

- Make a copy of the log file, and find the open file.
- If this isn't a `\usepackage`, back up until you find one.
- Do you have experience of LaTeX internals?
- No. This is a good time to seek expert advice. Go to `tex.sx` [4]. If the problem hasn't been reported, post a question. Be specific, and include your preamble and log.
- Yes. Figure out what the problem is. Check reports at `tex.sx` [4]. If it hasn't been reported, notify the package author.

This ends the discussion of problems that may occur in *your* files. The next section describes an actual problem of the author that took far too long to understand, and, in the end, wasn't really a "LaTeX problem", although that's where it reared its ugly head.

12 A real puzzlement

Once in a while, not even tracing can direct you to the solution of the problem.

Two facts are important:

- I live and work in the U.S. and my workstation is set up with (presumably appropriate) local defaults, i.e., ASCII.
- I compile from the command line, and don't use `\batchmode` or `\nonstopmode`.

What showed up on the screen:

```
Overfull \hbox (23.1113pt too wide) in paragraph at lines 3288--3301
\OML/cmm/m/it/10.95 A$\T1/ptm/m/n/10.95 , as in
```

The corresponding content of the log:

```
Overfull \hbox (23.1113pt too wide) in paragraph at lines 3288--3301
\OML/cmm/m/it/10.95 A$\T1/ptm/m/n/10.95 , as in Ÿ[], is a degree-
```

What was in the source file:

```
..., as in \S\ref{SS:changing}, is a degree-1 ...
```

Figure 1: A most puzzling problem

A few months ago, a file was persistently stalling before processing was finished, locking my screen. In order to regain control, it was necessary to open another session and kill the job. This allowed me to enter `ctrl-C` to the stalled session, to return to a prompt. The last thing shown on the screen was a partial report of an overfull box. Enough text was present to be able to locate the problem area in the source ... except that the source looked perfectly valid. (See figure 1.) There was, fortunately, a log produced, albeit incomplete.

After going through the steps described above, I managed to cut the file down to a single, brief, paragraph; if I removed anything more from the beginning of the paragraph, the error didn't occur. The problem appeared to be related to the overfull box. At this point, I sought help from someone with more systems knowledge than I have.

After looking closely at what was on the screen and what it corresponded to in the log, we noticed a "strange" character — Ÿ. (This is in position `"78` in the `cmsy` font, and is Unicode character U+0178.) Since I'm used to working with English texts, and only infrequently deal with accents, I'm not used to seeing non-ASCII characters, and certainly not in a log from an entirely English text. What was happening was that the physical environment I've been working in is not set up to recognize UTF-8 input, and the screen was freezing as a result.

The workaround I was given was to put this line into a file named `.i18n` in my home directory:

```
LANG="en_US.utf8"
```

This doesn't solve the problem entirely — the file still freezes the screen, but the job completes, and I can issue `ctrl-C` to regain a prompt. But figuring out the problem and how to get around it were sorely trying.

Sometimes what one thinks is a LaTeX bug isn't. Keep an open mind.

13 Oddments (post-conference additions)

There are some easily identified problems that occur frequently, but the source may not be generally known. This seems a worthwhile place to identify them.

- The `Missing character:` warning
  `There is no ; in font nullfont!`
  is almost invariably the result of a syntax error — a missing semicolon — in a `tikzpicture`. Other repeating punctuation characters citing `nullfont` may also be associated with some tikz expression.
- Similar warnings citing other fonts need to be researched. No line number is given in the log, but the number of the last completed page will be there. Compare the input with the output to see what is missing.

14 Acknowledgments

Thanks to GUST for hosting TUG'17 at Bachotek along with their annual meeting, and thanks to the participants whose questions after my talk provided interesting and helpful new ideas.

References and resources

[1] Victor Eijkhout, *TEX by Topic: A TEXnician's Reference*, Addison-Wesley (U.K.), 1991. `eijkhout.net/texbytopic/texbytopic.html` or `texdoc texbytopic`.

[2] Donald E. Knuth, *The TEXbook*, Addison-Wesley, Reading, 1986.

[3] Leslie Lamport, *LATEX, A Document Preparation System*, 2nd edition, Addison-Wesley, Reading, 1994.

[4] `tex.stackexchange.com`, a question and answer forum with extensive archives.

◇ Barbara Beeton
American Mathematical Society
Providence, RI, USA
`bnb (at) ams dot org`