The

LaTeX String Equality Comparison Primer

by

慈夆流

Contents

1 Rationale 3

2 A Word of Caution 4

3 Setup 5

4 ifstrequal 6

5 ifdefstring 7

6 ifdefstrequal 8

7 ifx 9

8 eqTextEquals 10

1 Rationale

While there is a justified interest in the question what it really *is*, in essence, that distinguishes a Turing-complete programming language from, say, mere markup, sane pragmatics know very well that this not-so elusive bit lies in

- variables,
- conditionals (if/then/else...),
- loopage (loop, while, break...), and
- anonymous and named functions.

All of these four factors are (in principle) part of TEX. You certainly could do with less *in principle*, but *in practice* (read: most of the time), you want your idiom to possess all four factors, as program sources quickly become obfuscated and convoluted if any one of these is missing or poorly implemented.

In the present document, we are solely concerned with conditionals and branching, more specifically: how to write LATEX code that branches according to a string equality test.

Equality testing is the most fundamental of all tests, so fundamental that a programming language with a poor implementation of equality testing must be considered a poor programming language.

According to this metric, TEX (and LATEX) are poor programming languages indeed. Anyone who has heard of TEX's `\ifx` command, be it from the Book or from the Net, and tried to make it work and not commit a blunder that leads to mysterious, subtle and hard-to-debug errors can testify to this.

Forunately for us, the community that has grown up around TEX—and especially LATEX—has provided the world of free and open-source software with a plethora of packages that make many things much easier for Joe the average LATEX writer.

To the best of my knowledge, easy and straightforward string equality testing has sadly not yet been fully covered.

In this paper, i want to discuss the merits of the four most popular string equality commands that float around in forum discussions: these are `\ifdefstrequal`, `\ifdefstring`, and `\ifstrequal`, all from the `etoolbox` package, and TEX's 'primeval' (and confusingly named) `\ifx` command.

In the last section, i then propose a new macro by the name of `\eqTextEquals`, which combines the knowledge gained from documentation & testing, and makes string equality testing a no-brainer.

2 A Word of Caution

During the preparation of this document, an Error krept in, where i had forgotten to define macro `\defC`. Now we all know well that whenever you use an undefined name in your TEXsource, say:

```
\unknown
```

then the compiler will most certainly opt-out with

```
! Undefined control sequence.
l.73 \unknown
```

But not so when you're doing string comparison. Testing two texts for equality requires such deep and mysterious inner workings of TEXto be used that testing whether a given name is defined gets swept under the carpet somehow. So you *can* write this:

```
\ifdefstrequal{\cmdA}{\unknown}{if branch}{else branch}
```

and `\ifdefstrequal` will duly choose the else branch—which is correct, in a way, since an undefined value can never ever equal any defined value (though in JavaScript, this is not true). Still, this is an unexpected behavior, as the Principle of Least Surprise would dictate that undefined names *always* make TEX halt with error. So take special care you have all your marbles defined, and no typos in your string comparison arguments.

3 Setup

To perform the tests, the following definitions are presupposed:

```
\newcommand{\cmdA}{sometext}
\newcommand{\cmdB}{sometext}
\newcommand{\cmdC}{othertext}
\def\defA{sometext}
\def\defB{sometext}
\def\defB{othertext}
```

All tests are done in pairs that are similar to this one:

```
\ifstrequal{\cmdA}{sometext}{OK}{wrong}
\ifstrequal{\cmdA}{othertext}{wrong}{OK}
```

Both clauses of a pair do a string comparison using the command under scrutiny; the first one should execute the `if` clause, the second one the `else` clause of the statement. If a test succeeds, an `OK` is displayed; if it fails, `wrong` is printed out. Thus, a given argument pattern is usable with the given command if two `OK`s are printed behind the command quotations; in case the word `wrong` appears, the command is not suited for the kind of arguments used in that test pair—be it TEX `\def`s, LATEX `\newcommand`s, or string literals.

4 ifstrequal

`\ifstrequal`{⟨ *string* ⟩}{⟨ *string* ⟩}{⟨ *true* ⟩}{⟨ *false* ⟩}

☛ Compares two strings and executes ⟨ *true* ⟩ if they are equal, and ⟨ *false* ⟩ otherwise. The strings are not expanded in the test and the comparison is category code agnostic. Control sequence tokens in any of the ⟨ *string* ⟩ arguments will be detokenized and treated as strings. This command is robust.

etoolbox.pdf

`\ifstrequal{sometext}{sometext}`: OK
`\ifstrequal{sometext}{othertext}`: OK

`\ifstrequal{\cmdA}{sometext}`: wrong
`\ifstrequal{\cmdA}{othertext}`: OK

`\ifstrequal{\defA}{sometext}`: wrong
`\ifstrequal{\defA}{othertext}`: OK

`\ifstrequal{\cmdA}{\cmdB}`: wrong
`\ifstrequal{\cmdA}{\cmdC}`: OK

`\ifstrequal{\cmdA}{\defB}`: wrong
`\ifstrequal{\cmdA}{\defC}`: OK

`\ifstrequal{\defA}{\cmdB}`: wrong
`\ifstrequal{\defA}{\cmdC}`: OK

`\ifstrequal{\defA}{\defB}`: wrong
`\ifstrequal{\defA}{\defC}`: OK

Verdict.—A jolly useless command as such, as it merely checks whether two literal texts are equal. In many years of programming, i've virtually *never* felt the urge to litter my sources with non-consequential incantations à la `42 == 42`. Still, it's useful inside macro definitions, as shown in the definition of `\eqTextEquals`, below.

Advice.—You never want to use this one, except when writing a macro yourself.

5 ifdefstring

`\ifdefstring`{⟨ *command* ⟩}{⟨ *string* ⟩}{⟨ *true* ⟩}{⟨ *false* ⟩}

☛ Compares the replacement text of a ⟨ *command* ⟩ to a ⟨ *string* ⟩ and executes ⟨ *true* ⟩ if they are equal, and ⟨ *false* ⟩ otherwise. Neither the ⟨ *command* ⟩ nor the ⟨ *string* ⟩ is expanded in the test and the comparison is category code agnostic. Control sequence tokens in the ⟨ *string* ⟩ argument will be detokenized and treated as strings. This command is robust. Note that it will only consider the replacement text of the ⟨ *command* ⟩.

etoolbox.pdf

`\ifdefstring{\cmdA}{sometext}`: OK
`\ifdefstring{\cmdA}{othertext}`: OK

`\ifdefstring{\defA}{sometext}`: OK
`\ifdefstring{\defA}{othertext}`: OK

`\ifdefstring{\cmdA}{\cmdB}`: wrong
`\ifdefstring{\cmdA}{\cmdC}`: OK

`\ifdefstring{\cmdA}{\defB}`: wrong
`\ifdefstring{\cmdA}{\defC}`: OK

`\ifdefstring{\defA}{\cmdB}`: wrong
`\ifdefstring{\defA}{\cmdC}`: OK

`\ifdefstring{\defA}{\defB}`: wrong
`\ifdefstring{\defA}{\defC}`: OK

Verdict.—Use this command to compare a TEX `\def` or LATEX `\newcommand` expansion with a literal text. If the second argument is or contains a macro, only the `else` clause will be executed.

`\ifdefstring` should prove useful in a lot of circumstances. The fact that the first argument can be both a `\def` or a `\newcommand` makes it robust (in the sense that you don't need to know which command was defined in what way); the fact that the second argument must be a literal helps to keep the source simple and readable.

Advice.—Use `\ifdefstring` especially in cases where a macro is a state-keeping variable that operates on short keywords.

6 ifdefstrequal

`\ifdefstrequal`{⟨ *command* ⟩}{⟨ *command* ⟩}{⟨ *true* ⟩}{⟨ *false* ⟩}

☛ Performs a category code agnostic string comparison of the replacement text of two commands. This command is similar to `\ifdefstring` except that both arguments to be compared are macros. This command is robust.

etoolbox.pdf

`\ifdefstrequal{\cmdA}{sometext}`: wrong
`\ifdefstrequal{\cmdA}{othertext}`: OK

`\ifdefstrequal{\defA}{sometext}`: wrong
`\ifdefstrequal{\defA}{othertext}`: OK

`\ifdefstrequal{\cmdA}{\cmdB}`: OK
`\ifdefstrequal{\cmdA}{\cmdC}`: OK

`\ifdefstrequal{\cmdA}{\defB}`: OK
`\ifdefstrequal{\cmdA}{\defC}`: OK

`\ifdefstrequal{\defA}{\cmdB}`: OK
`\ifdefstrequal{\defA}{\cmdC}`: OK

`\ifdefstrequal{\defA}{\defB}`: OK
`\ifdefstrequal{\defA}{\defC}`: OK

Verdict.—Works as advertised.

Advice.—Use `\ifdefstrequal` to ckeck whether two macros have the same string expansion.

7 ifx

`\ifx`⟨ *command* ⟩⟨ *command* ⟩⟨ *true* ⟩⟨ *false* ⟩`\fi`

With the `\ifx` command you can perform conditional compilation by testing for macro equivalence. It does not expand the given macros. The two macros are considered equal if

- both are macros and
- the first level expansion is equal and
- they have the same state with regards to `\long` and `\outer`.

http://en.wikibooks.org/wiki/TeX/ifx

`\ifx\cmdA\cmdB` OK
`\ifx\cmdA\cmdC` OK

`\ifx\cmdA\defB` wrong
`\ifx\cmdA\defC` OK

`\ifx\defA\cmdB` wrong
`\ifx\defA\cmdC` OK

`\ifx\defA\defB` OK
`\ifx\defA\defC` OK

Verdict.—This macro's description hurts my feelings. Seriously. According to the tests, it would *appear* to work correctly when both arguments are either TEX `\def`s or else LATEX `\newcommand`s, but be aware that in the general case you will *not* know for sure whether a given `\x` was defined in whatever which way (unless you did it yourself a short while ago or you perused the sources). The restriction regarding `\long` and `\outer` makes `\ifx` even more brittle and ultimately useless for anyone but the most hard-core TEXnocrats.

Advice.—Use with care and only after testing whether the command works for your special case. Consider to use `\ifdefstrequal` instead, as that one offers fewer opportunities to write buggy code. Also, the syntax of `\ifdefstrequal` is more in keeping with LATEXconventions.

8 eqTextEquals

`\eqTextEquals`{⟨ *valueOne* ⟩}{⟨ *valueTwo* ⟩}{⟨ *true* ⟩}{⟨ *false* ⟩}

☛ Uses the `\ifdefmacro` command from the `\etoolbox` package to test whether ⟨ *valueOne* ⟩ and ⟨ *valueTwo* ⟩ are macros or literals; based on this knowledge, it decides whether to use `\ifdefstrequal`, `\ifdefstring`, or `\ifstrequal` to perform a string comparison.

`\eqTextEquals{sometext}{sometext}`: OK
`\eqTextEquals{sometext}{othertext}`: OK

`\eqTextEquals{\cmdA}{sometext}`: OK
`\eqTextEquals{\cmdA}{othertext}`: OK

`\eqTextEquals{\defA}{sometext}`: OK
`\eqTextEquals{\defA}{othertext}`: OK

`\eqTextEquals{\cmdA}{\cmdB}`: OK
`\eqTextEquals{\cmdA}{\cmdC}`: OK

`\eqTextEquals{\cmdA}{\defB}`: OK
`\eqTextEquals{\cmdA}{\defC}`: OK

`\eqTextEquals{\defA}{\cmdB}`: OK
`\eqTextEquals{\defA}{\cmdC}`: OK

`\eqTextEquals{\defA}{\defB}`: OK
`\eqTextEquals{\defA}{\defC}`: OK

Verdict.—Works as advertised.

Advice.—Use `eqTextEquals` to replace all the other commands discussed in this document—it's much simpler, more versatile, less surprising, and puts less strain onto your name memory.

Implementation

The `\eqTextEquals` macro has a fairly simple implementation, *viz.*:

```
\newcommand{\eqTextEquals}[4]{
  \ifdefmacro{#1}{
    % #1 is a macro
    \ifdefmacro{#2}{
      % #1 and #2 are macros
      \ifdefstrequal{#1}{#2}{#3}{#4}
    }{
      % #1 is a macro, #2 is a literal
      \ifdefstring{#1}{#2}{#3}{#4}
      }
  }{
    % #1 is a literal
    \ifdefmacro{#2}{
      % #1 is a literal, #2 is a macro
      \ifdefstring{#2}{#1}{#3}{#4}
    }{
      % #1 and #2 are literals
      \ifstrequal{#1}{#2}{#3}{#4}
      }
    }
  }

\newcommand{\eqIfTextEqualsThen}[3]{
  \eqTextEquals{#1}{#2}{#3}{}
  }

\newcommand{\eqUnlessTextEqualsThen}[3]{
  \eqTextEquals{#1}{#2}{}{#3}
  }
```