The LaTeX verifycommand Package

v1.00 — 2024/01/11

© 2024 Brian Dunn
https://github.com/bdtc/verifycommand

Verifies definitions have not changed.

Abstract

For package authors who patch code from other packages.

To improve reliability, the verifycommand package provides a way to verify that macros or environments have not changed. This allows a package author to check before patching a definition. If a definition is not as expected, a warning is issued. At the end of the compile, a list of all changed definitions is displayed.

Contents

1 Introduction

Patching a macro or environment from another package risks the possibility that the other author has made an update and changed something unexpected, breaking your own package when it tries to apply the patch.

The traditional way to check a definition you wish to modify is to copy the expected definition into your package under a new name, then compare to see if the current definition is the same as it was when your package was first created. For a few definitions this may work well, but as the number of patches goes up things get more and more unwieldy.

The verifycommand package uses MD5 checksums instead of copying entire definitions. If something has changed, a warning is issued telling the name of the defintion, and optionally telling the name of your own package and the package being modified. This improves code reliability, and allows package authors to get an early warning when an author of some other package has made an unexpected change.

In many cases, the patch or replacement may still function correctly even when the original has changed in some way. For this reason, only a warning is issued, not an error.

2 How it works

`\VerifyCommand` and `\VerifyEnvironment` are used to test whether a definition has changed. Each definition is given an MD5 checksum, which is compared to the expected checksum given as arguments of `\VerifyCommand` and `\VerifyEnvironment`. If a checksum does not match, a warning is issued, flagging the definition for attention.

The MD5 checksum is of the text of the code part of the underlying definition, as it would be displayed by the `\meaning` command. For environments, the end code is checked separately. The check detects changes in the replacement text of the definition, and may or may not detect changes in the number or type of parameters, `\long`, or type of robustness, depending on the type of definition. Some definitions may have the same checksum if they have the same replacement code but different argument types, for example a `\NewDocumentCommand` with two mandatory arguments vs another with one optional and one mandatory, if they both have the same replacement code.

When something does not match, the current checksum is printed to the terminal, and the author may copy/paste that into the parameter of `\VerifyCommand` or `\VerifyEnvironment` to update the expected values.

3 How to use verifycommand

3.1 The user interface

`\VerifyCommand` [⟨*yourpackagename*⟩] [⟨*theirpackagename*⟩] {⟨`\macroname`⟩} {⟨MD5 *checksum*⟩}

`\VerifyEnvironment` [⟨*yourpackagename*⟩] [⟨*theirpackagename*⟩] {⟨`envname`⟩} {⟨*begin* MD5 *checksum*⟩} {⟨*end* MD5 *checksum*⟩}

Use one of these macros just before patching a macro or environment, as seen below.

Note that there is one checksum for `\VerifyCommand`, but there are two checksums for `\VerifyEnvironment`: once for the begin section and one for the end section.

If there is only one optional argument, it is used as your package name.

3.2 Placing the macros

When first using verifycommand, use empty checksums, placing `\VerifyCommand` or `\VerifyEnvironment` before each place where something gets patched. This is probably not required where things are entirely replaced, or prepended or appended.

```
\VerifyCommand{\LaTeX}{}
<patch \LaTeX here>

\VerifyCommand[mypackage]{\textcolor}{}
<patch \textcolor here>

\VerifyCommand[mypackage][graphics]{\rotatebox}{}
<patch \rotatebox here>

\VerifyEnvironment{tabbing}{}{}
<patch tabbing here>
```

3.3 Finding the checksums of the current definitions

In the above examples, testing `\LaTeX` would print a warning showing the correct MD5 checksum. Testing `\textcolor` would do the same, but as a `\PackageWarning` from mypackage. Testing `\rotatebox` would also mention the package being tested, graphics. Testing `tabbing` issues a separate warning for the begin and end sections.

```
Warning: Something may fail. A definition has changed:
        \LaTeX
        FAAAC6146C9A80F46A1F029B67923851
         on input line 464.
```

```
Package mypackage Warning: Something may fail. A definition has changed:
(mypackage)                \textcolor
(mypackage)                E1E2B5A908AA1BCDDF6BEA038596A381
(mypackage)                 on input line 465.

Package mypackage Warning: Something may fail. A definition has changed:
(mypackage)                graphics: \rotatebox
(mypackage)                2472999B02C97AC847128AF24C55D150
(mypackage)                 on input line 466.

Warning: Something may fail. A definition has changed:
        tabbing
        1AD73B4527AD30969CF3219F2FB1306B
         on input line 518.

Warning: Something may fail. A definition has changed:
        (end)tabbing
        E8326AC43EE0A6E922A20F2A798BD177
         on input line 518.
```

And at the end of the compile, a summary is given:

```
--------------------------------
WARNING: Something may fail.
Definition changed: \LaTeX
Definition changed:    FAAAC6146C9A80F46A1F029B67923851
Definition changed: \textcolor
Definition changed:    E1E2B5A908AA1BCDDF6BEA038596A381
Definition changed: graphics: \rotatebox
Definition changed:    2472999B02C97AC847128AF24C55D150
Definition changed: tabbing
Definition changed:    1AD73B4527AD30969CF3219F2FB1306B
Definition changed: (end)tabbing
Definition changed:    E8326AC43EE0A6E922A20F2A798BD177
--------------------------------
```

3.4 Assigning the checksums

Copy the checksums from the warnings messages into the source. When this is done, there are no more verifycommand warnings unless one of these defintions changes:

```
\VerifyCommand{\LaTeX}{FAAAC6146C9A80F46A1F029B67923851}
<patch \LaTeX here>

\VerifyCommand[mypackage]{\textcolor}
                 {E1E2B5A908AA1BCDDF6BEA038596A381}
<patch \textcolor here>
```

```
\VerifyCommand[mypackage][graphics]{\rotatebox}
                {2472999B02C97AC847128AF24C55D150}
<patch \rotatebox here>

\VerifyEnvironment{tabbing}
     {1AD73B4527AD30969CF3219F2FB1306B}%   beginning code
     {E8326AC43EE0A6E922A20F2A798BD177}%   endind code
<patch tabbing here>
```

3.5 When a definition is changed

When something being verified changes at a later time, the resulting warning will let the user know that the patches may not work as expected. Because the test is done before the patch, this warning will be issued before the patch is even attempted.

When testing many packages in bulk, a utility such as **grep** can report the changed definitions. Search the log file for “Definition changed:”.

3.6 Disabling the package

verifycommand relies on knowing the internal structure of various kinds of definitions. It is possible that these may change some day, causing endless warnings for that kind of definition. Should that happen, it will be necessary to disable the verifycommand package until it can be updated. Use the disable option to do so.

```
\usepackage[disable]{verifycommand}
```

4 Code

4.1 Package requirements

```
\RequirePackage{etoolbox}
\RequirePackage{iftex}
```

4.2 Package options

Package option to disable all functions.

VERCMD@enable (*bool*) Is the package enabled?

```
\newbool{VERCMD@enable}
\booltrue{VERCMD@enable}
```

disable (*Opt*) Turn off all functions.

```
\DeclareOption{disable}{%
    \boolfalse{VERCMD@enable}%
    \typeout{----}%
    \typeout{Package verifycommand: Turned off by option 'disable'.}%
    \typeout{----}%
}

\ProcessOptions\relax
```

4.3 Support macros

\VERCMD@backslash The literal \ character.

This is used later because some internal definitions use double \\ as part of their name.

```
\catcode`\&=0
&catcode`&\=12
&def&VERCMD@backslash{\}
&catcode`&\=0
\catcode`\&=4
```

4.4 MD5 hashing

The MD5 hash is used for `lateximage` filenames for SVG math.

```
\newcommand{\VERCMD@mdfivesum}[1]{%
    \PackageError{verifycommand}
        {No MD5 macro was found}
        {%
            Verifycommand must find the macros \protect\pdfmdfivesum\space
            or \protect\mdfivesum.%
        }
}
```

The default for PDF LaTeX, DVI LaTeX, upLaTeX, etc:

```
\ifdef{\pdfmdfivesum}
    {\let\VERCMD@mdfivesum\pdfmdfivesum}
    {}
```

For LuaLaTeX:

```
\ifLuaTeX
\RequirePackage{pdftexcmds}
\let\VERCMD@mdfivesum\pdf@mdfivesum
\fi
```

For XeLaTeX:

```
\ifXeTeX
\@ifundefined{pdffivesum}{}
    {\let\VERCMD@mdfivesum\pdfmdfivesum}
\@ifundefined{mdfivesum}{}
    {\let\VERCMD@mdfivesum\mdfivesum}
\fi
```

`\VERCMD@mdfive` {⟨\macroname⟩} Compute MD5 checksum, store in `\VERCMD@temp`.

```
\def\VERCMD@mdfive#1{%
    \edef\VERCMD@temp{\VERCMD@mdfivesum{\meaning#1}}%
}
```

4.5 Issuing warnings

`\VERCMD@whatchanged` Accumulates a list of changed definitions.

```
\newcommand*{\VERCMD@whatchanged}{}
```

`\VERCMD@addchanged` {⟨MD5*sum*⟩} {⟨*text*⟩} Add to the list of changed definitions.

```
\newcommand*{\VERCMD@addchanged}[2]{%
    \ifdefempty{\VERCMD@whatchanged}%
        {}%
        {\apptocmd{\VERCMD@whatchanged}{^^J}{}{}}%
    \apptocmd{\VERCMD@whatchanged}{Definition changed: #2^^J}{}{}%
   \apptocmd{\VERCMD@whatchanged}{Definition changed: \space\space\space#1}{}{}%
}
```

When the compile is finished, print the accumulated list of changed definitions.

```
\AfterEndDocument{
    \ifdefempty{\VERCMD@whatchanged}{}{%
        \typeout{--------------------------------}%
        \typeout{WARNING: Something may fail.}%
        \typeout{\VERCMD@whatchanged}%
        \typeout{--------------------------------}%
    }
}
```

`\VERCMD@ProgWarning` {⟨*text*⟩} Warning without a package name.

```
\def\VERCMD@ProgWarning#1{%
    \GenericWarning{%
%        (\jobname)\@spaces\@spaces%
        \@spaces\@spaces
    }{%
        Warning: #1%
    }%
}
```

`\VERCMD@Warning` {⟨*yourpackage*⟩} {⟨*theirpackage*⟩} {⟨*defn name*⟩}

If no package names, print a general warning. If package names are given, print a `\PackageWarning`.

```
\newcommand*{\VERCMD@Warning}[3]{%
    \ifblank{#1}%
        {%
            \VERCMD@ProgWarning{%
                Something may fail.  A definition has changed:\MessageBreak
                \ifblank{#2}{}{#2: }\string#3\MessageBreak
                \VERCMD@temp\MessageBreak
            }
          \expandafter\VERCMD@addchanged\expandafter{\VERCMD@temp}{\string#3}%
        }%
        {%
            \PackageWarning{#1}{%
                Something may fail.  A definition has changed:\MessageBreak%
                \ifblank{#2}{}{#2: }%
                \string#3\MessageBreak%
                \VERCMD@temp\MessageBreak%
            }%
        \expandafter\VERCMD@addchanged\expandafter{\VERCMD@temp}{\ifblank{#2}{}{#2: }\string#3}%
        }%
}
```

```
\ExplSyntaxOn
```

4.6 User interface

`\VerifyCommand` [⟨*yourpackage*⟩] [⟨*theirpackage*⟩] {⟨\commandname⟩} {⟨MD5 *checksum*⟩}

Test for various kinds of definitions, and convert them to MD5 checksums.

```
\NewDocumentCommand{\VerifyCommand}{O{} O{} m m}{%
```

Only if the package is enabled:

```
    \ifbool{VERCMD@enable}{%
```

Default to an un-detected definition type:

```
    \edef\VERCMD@temp{Unknown~definition}%
```

For `\NewDocumentCommand`, the macro name is "`\name code`" with a space in the middle.

```
%    % NewDocumentCommand:
     \ifcsdef{\cs_to_str:N #3~code}%
        {%
            \expandafter\VERCMD@mdfive%
                \csname \cs_to_str:N #3~code\endcsname%
        }%
        {%
```

For `\DeclareRobustCommand` with an optional argument, the macro name is "`\\name `" with a two backslashes and a trailing space.

```
%            % DeclareRobustCommmand with option:
             \ifcsdef{\VERCMD@backslash\cs_to_str:N #3~}%
                {%
                    \expandafter\VERCMD@mdfive%
                        \csname \cs_to_str:N #3~code\endcsname%
                }%
                {%
```

For `\DeclareRobustCommand`, the macro name is "`\name `" with a trailing space.

```
%                    % DeclareRobustCommand:
                     \ifcsdef{\cs_to_str:N #3~}%
                        {%
                            \expandafter\VERCMD@mdfive%
                                \csname \cs_to_str:N #3~\endcsname%
                        }%
                        {%
```

For `\newcommand` with an option, the macro name is "`\\name`", with two backslashes.

```
%                            % newcommand w/ option:
                             \ifcsdef{\VERCMD@backslash\cs_to_str:N #3}%
```

```
                                {%
                                    \expandafter\VERCMD@mdfive%
                        \csname \VERCMD@backslash\cs_to_str:N #3\endcsname%
                                }%
```

For `\newcommand`, the macro name is “`\name`”.

If none match, the default unknown definition warning is show in place of the checksum.

```
                                {%
%                                    % newcommand:
                                    \ifdef{#3}%
                                       {\VERCMD@mdfive#3}%
                                       {}%
                                }%
                        }%
                }%
        }%
```

If the checksum does not match the expected value, issue a warning.

```
    \ifdefstring{\VERCMD@temp}{#4}%
        {}%
        {%
            \VERCMD@Warning{#1}{#2}{#3}%
        }%
    }% if package enabled
    {}% if package not enabled
}
```

`\VerifyEnvironment` [⟨*yourpackage*⟩] [⟨*theirpackage*⟩] {⟨\commandname⟩} {⟨*begin* MD5 *checksum*⟩} {⟨*end* MD5 *checksum*⟩}

Test both the begin and end section of the environment.

```
\NewDocumentCommand{\VerifyEnvironment}{O{} O{} m m m}{%
```

Only if the package is enabled:

```
    \ifbool{VERCMD@enable}{%
```

Default to an un-detected definition type:

```
    \edef\VERCMD@temp{Unknown~definition}%
```

For `\NewDocumentEnvironment`, the macro name is “`\environment name code`” with internal spaces.

```
%     % NewDocumentEnvironment:
     \ifcsdef{environment~#3~code}%
        {%
```

```
            \expandafter\VERCMD@mdfive%
                \csname environment~#3~code\endcsname%
        }%
        {%
```

For `\newenvironment` with an optional argument, the macro name is “`\\name`”, with two backslashes.

```
%            % newenvironment with option:
             \ifcsdef{\VERCMD@backslash#3}%
                {%
                    \expandafter\VERCMD@mdfive%
                        \csname \VERCMD@backslash#3\endcsname%
                }%
                {%
```

For `\newenvironment`, the macro name is “`\name`”.

```
%                    % newenvironment:
                     \ifcsdef{#3}%
                        {\expandafter\VERCMD@mdfive\csname #3\endcsname}%
                        {}%
                }%
        }%
```

If the checksum does not match the expected value, issue a warning.

```
    \ifdefstring{\VERCMD@temp}{#4}%
        {}%
        {%
            \VERCMD@Warning{#1}{#2}{#3}%
        }%
```

Reset the default to an un-detected definition type:

```
    \edef\VERCMD@temp{Unknown~definition}%
```

For `\NewDocumentEnvironment`, the ending macro name is “`\environment name end aux `”, with spaces and a trailing space.

```
%    % end DocumentEnvironment:
     \ifcsdef{environment~#3~end~aux~}%
        {%
            \expandafter\VERCMD@mdfive
                \csname environment~#3~end~aux~\endcsname%
        }%
        {%
```

For `\newenvironment`, the ending macro name is “`\endname`”.

```
%            % end newenvironment:
             \ifcsdef{end#3}%
```

```
                {%
                    \expandafter\VERCMD@mdfive%
                        \csname end#3\endcsname%
                }%
                {}%
        }%
```

If the checksum does not match the expected value, issue a warning.

```
    \ifdefstring{\VERCMD@temp}{#5}%
        {}%
        {%
            \VERCMD@Warning{#1}{#2}{(end)#3}%
        }%
    }% if package enabled
    {}% if package not enabled
}
```

```
\ExplSyntaxOff
```

5 verifycommand package maintenance

To compile `verifycommand.sty` and `\verifycommand.pdf` from `verifycommand.dtx` and `verifycommand.ins`:

```
pdflatex verifycommand.ins
pdflatex verifycommand.dtx
pdflatex verifycommand.dtx
makeindex -s gglo.ist -o verifycommand.gls verifycommand.glo
splitindex verifycommand.idx -- -s gind.ist
pdflatex verifycommand.dtx
pdflatex verifycommand.dtx
```

Change History

v1.00

General: 2024/01/11 Initial version. . 1

Index of Objects

This is an index of macros, environments, booleans, counters, lengths, packages, classes, options, keys, files, and various other programming objects. Each is listed by itself, and also by category. In some cases, they are further subdivided by [class].

Numbers written in italic refer to the page where the corresponding entry is described; numbers underlined refer to the code line of the definition.