The LaTeX verifycommand Package

v1.10 — 2024/09/03

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 since the patches were last designed. This may be checked before applying the patch. 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.

Conditionals are provided, allowing multiple versions of a definition to be tested and patched, warning if no known version is found. Conditionals also allow the package to verify that several macros are unchanged before taking a single common action.

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 Warnings

The document *LaTeX for authors* (**`texdoc usrguide`** recommends the use of LaTeX hooks where possible (**`texdoc lthooks`**).

⚠ kernel changes

Commands created with \NewDocumentCommand and related may change their definitions when the inner workings of \NewDocumentCommand are optimized. When this occurs, any checksums for these commands will change, even though the associated patches probably will continue to work. If you use verifycommand for these commands, expect the checksums to need updates at some point in the future.

Use \NewCommandCopy and \NewEnvironmentCopy to copy commands for reuse. These work with all kinds of commmand definitions. See **texdoc usrguide** for details.

Use \ShowCommand or \ShowEnvironment to see an existing definition, which again works with all forms of commands.

4 How to use verifycommand

4.1 The user interface

In the following, [⟨*yourpackagename*⟩] is the package doing the testing and patching, and [⟨*theirpackagename*⟩] is the package which defines the original macro. Using these optional package names make it easier to find where to make changes if needed.

⚠ optional arguments

If there is only one optional argument, it is used as [⟨*yourpackagename*⟩], to identify where to find the \VerifyCommand checksum entry. The second optional argument identifies what is being patched, in case it has been changed.

\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. A test is performed to see if the definition is as expected before applying a patch.

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

It may be necessary to test for each of several possible versions of a definition, then patch accordingly. The following test and apply a true or false clause, without issuing a warning.

\IfVerifyCommand [⟨*yourpackagename*⟩] [⟨*theirpackagename*⟩] {⟨\macroname⟩}
{⟨MD5 *checksum*⟩}
{⟨*true*⟩} {⟨*false*⟩}

\IfVerifyEnvironmentBegin [⟨*yourpackagename*⟩] [⟨*theirpackagename*⟩] {⟨envname⟩}
{⟨*begin* MD5 *checksum*⟩}
{⟨*true*⟩} {⟨*false*⟩}

`\IfVerifyEnvironmentEnd` `[`⟨*yourpackagename*⟩`]` `[`⟨*theirpackagename*⟩`]` `{`⟨envname⟩`}`
`{`⟨*end* MD5 *checksum*⟩`}`
`{`⟨*true*⟩`}` `{`⟨*false*⟩`}`

Tests may be done as a group, either to try several versions of the same definition, or to verify several definitions and act if any of them fail:

`\TestVerifyCommands` Starts a group of tests.

`\IfVerifyCommandPassed` `{`⟨*true*⟩`}` `{`⟨*false*⟩`}` Act if any of a group passed. May be used to try to patch multiple versions of a command.

`\IfVerifyCommandFailed` `{`⟨*true*⟩`}` `{`⟨*false*⟩`}` Act if any of a group failed. May be used to verify several definitions are unchanged.

A warning may be also issued:

`\VERCMDWarning` `{`⟨*yourpackage*⟩`}` `{`⟨*theirpackage*⟩`}` `{`⟨*defn name*⟩`}` Used to issue a warning that the macro has changed.

See section 4.6 for examples of conditional tests.

4.2 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)
```

4.3 Finding the checksums of the current definitions

The warnings are issued and the correct checksums are given. The type of warning depends on the usage:

- Verifying `\LaTeX` would print a warning showing the correct MD5 checksum.

```
Warning: A definition has changed:
        \LaTeX
```

```
(FAAAC6146C9A80F46A1F029B67923851)
 on input line 464.
```

- Verifying `\textcolor` would do the same, but as a `\PackageWarning` from mypackage, which is the package doing the testing.

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

- Verifying `\rotatebox` would also issue a warning from mypackage, but also mention the package being tested, graphics.

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

- Verifying `tabbing` issues a separate warning for the begin and end sections.

```
Warning: A definition has changed:
        tabbing
        (1AD73B4527AD30969CF3219F2FB1306B)
         on input line 518.

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

At the end of the compile, a summary is given:

```
--------------------------------
Package verifycommand Warning: Definitions have changed.
Patches for the following macros may need to be updated.
See the previous warnings for line numbers.

Syntax:  Patching pkg -> Defining pkg: Macro (checksum)
-----
(verifycommand)  \LaTeX
                   (FAAAC6146C9A80F46A1F029B67923851)
(verifycommand)  mypackage -> \textcolor
                   (E1E2B5A908AA1BCDDF6BEA038596A381)
(verifycommand)  mypackage -> graphics: \rotatebox
                   (2472999B02C97AC847128AF24C55D150)
(verifycommand)  tabbing
                   (1AD73B4527AD30969CF3219F2FB1306B)
(verifycommand)  (end)tabbing
                   (E8326AC43EE0A6E922A20F2A798BD177)
--------------------------------
```

4.4 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)
```

4.5 When a definition has 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 which macros have changed. Search the log file for "`(verifycommand)`".

4.6 Testing for changed macros

To test to see if a macro or environment has changed, use `\IfVerifyCommand`, `\IfVerifyEnvironmentBegin`, or `\IfVerifyEnvironmentEnd`.

Example:

```
\IfVerifyCommand[mypackage][LaTeX kernel]{\LaTeX}{123}{True}{False}
```

Result: False

Example:

```
\IfVerifyCommand[mypackage][LaTeX kernel]{\LaTeX}
      {FAAAC6146C9A80F46A1F029B67923851}
      {True}{False}
```

Result: True (Assuming \LaTeX has not actually changed since this manual was generated.)

Example:

```
\IfVerifyEnvironmentEnd[mypackage][LaTeX kernel]{tabbing}
      {E8326AC43EE0A6E922A20F2A798BD177}
      {True}{False}
```

Result: True

If the verifycommand package is disabled, these tests will always return true.

These tests do not issue warnings if the test fails, so they may be used to test against several possible definitions, chosing the appropriate patch depending on which match is found.

To try to patch multiple versions of the same command, and issue a warning if no match is found:

```
\TestVerifyCommands

\IfVerifyCommand[mypackage][theirpackage]{\theirmacro}
      {123123123123123123123123123123}
      {(Patch an older version of the macro.)}

\IfVerifyCommand[mypackage][theirpackage]{\theirmacro}
      {456456456456456456456456456456}
      {(Patch a newer version of the macro.)}

\IfVerifyCommandPassed
      {}% One of the patches worked.
      {\VERCMDWarning{mypackage}{theirpackage}{\theirmacro}}
```

To verify several commands have not changed:

```
\TestVerifyCommands

\VerifyCommand[mypackage][theirpackage]{\firstmacro}
      {789789789789789789789789789789}
\VerifyCommand[mypackage][theirpackage]{\secondmacro}
      {272727272727272727277272727272}
\VerifyCommand[mypackage][theirpackage]{\thirdmacro}
      {848384838483848384838483848384838483}

\IfVerifyCommandFailed
  {Something is not correct.  (Warnings have been issued above.)}
  {}% All are fine.
```

4.7 Disabling the package

verifycommand relies on knowing the internal structure of various kinds of definitions. It is possible that these may change, 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}
```

Package warnings will stop, and the conditional tests will always return true.

If the package is disabled, the boolean `VERCMDenable` will be `false`. This may be used to help decide which patches to apply as a default.

5 Code

5.1 Package requirements

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

5.2 Package options

Package option to disable all functions.

`VERCMDenable` (*bool*) Is the package enabled?

```
\newbool{VERCMDenable}
\booltrue{VERCMDenable}
```

`disable` (*Opt*) Turn off all functions.

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

\ProcessOptions\relax
```

5.3 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
```

5.4 MD5 hashing

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

The default definition if no MD5 function is found. This will be changed below if an MD5 function is available.

```
\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}}%
}
```

5.5 Issuing warnings

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

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

`VERCMD@this@ltxcmd` (*bool*) True if this command is defined by `\NewDocumentCommand` or related.

```
\newbool{VERCMD@this@ltxcmd}
\boolfalse{VERCMD@this@ltxcmd}
```

`VERCMD@failed@ltxcmd` (*bool*) True if any command that failed was defined by `\NewDocumentCommand` or related.

```
\newbool{VERCMD@failed@ltxcmd}
\boolfalse{VERCMD@failed@ltxcmd}
```

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

```
\newcommand*{\VERCMD@addchanged}[2]{%
```

Newline control for pretty print.

```
    \ifdefempty{\VERCMD@whatchanged}%
        {}%
        {\apptocmd{\VERCMD@whatchanged}{^^J}{}{}%
```

ID the message as from verifycommand, add the text, add the checksum.

```
    \apptocmd{\VERCMD@whatchanged}{%
        (verifycommand)\space\space#2^^J%
        \space\space\space\space\space\space\space\space\space\space%
        \space\space\space\space\space\space\space\space\space(#1)%
    }{}{}%
```

Optionally add a * after the checksum if the command was defined with `\NewCommand` or related.

```
    \ifbool{VERCMD@this@ltxcmd}{%
        \apptocmd{\VERCMD@whatchanged}{ *}{}{}%
        \booltrue{VERCMD@failed@ltxcmd}%
    }{}%
    \boolfalse{VERCMD@this@ltxcmd}%
}
```

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

```
\AtEndDocument{
    \ifdefempty{\VERCMD@whatchanged}{}{%
        \typeout{--------------------------------}%
        \typeout{Package verifycommand Warning: Definitions have changed.}%
        \typeout{Patches for the following macros may need to be updated.}%
        \typeout{See the previous warnings for line numbers.}%
        \typeout{}%
     \typeout{Syntax:\space\space Patching pkg -> Defining pkg: Macro (checksum)}%
        \typeout{-----}%
        \typeout{\VERCMD@whatchanged}
        \typeout{-----}%
        \typeout{Look for updates to these packages.}
%
        \ifbool{VERCMD@failed@ltxcmd}{
     \typeout{Any of the above marked with a * may be due to changes in LaTeX internals,}
        \typeout{the most recently known of which was 2023/12/01.}
        \typeout{If so, look for updates for the LaTeX sytem as well.}
        }{}
%
        \typeout{--------------------------------}
    }
}
```

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

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

`\VERCMDWarning` {⟨*yourpackage*⟩} {⟨*theirpackage*⟩} {⟨*defn name*⟩}

If no package names, print a general warning. If package names are given, print a `\PackageWarning`. Either way, also add to the summary report.

```
\newcommand*{\VERCMDWarning}[3]{%
    \ifblank{#1}%
        {%
            \VERCMD@ProgWarning{%
                A definition has changed:\MessageBreak
                \ifblank{#2}{}{#2: }\string#3\MessageBreak
                (\VERCMD@temp)\MessageBreak
            }%
            \expandafter\VERCMD@addchanged%
                \expandafter{\VERCMD@temp}{\string#3}%
        }%
        {%
            \PackageWarning{#1}{%
                A definition has changed:\MessageBreak%
                \ifblank{#2}{}{#2: }%
                \string#3\MessageBreak%
                (\VERCMD@temp)\MessageBreak%
            }%
            \expandafter\VERCMD@addchanged%
              \expandafter{\VERCMD@temp}{#1 -> \ifblank{#2}{}{#2: }\string#3}%
        }%
}

\ExplSyntaxOn
```

5.6 User interface

`VERCMD@passed` (*bool*) True if any test passed.

```
\newbool{VERCMD@passed}
```

`VERCMD@failed` (*bool*) True if any test failed.

```
\newbool{VERCMD@failed}
```

`\TestVerifyCommands` Starts a new set of `\VerifyCommand` tests, after which the booleans will tell if any passed and if any failed.

```
\newcommand*{\TestVerifyCommands}
{
    \boolfalse{VERCMD@passed}
    \boolfalse{VERCMD@failed}
}
```

`\IfVerifyCommandPassed` `{`⟨*true*⟩`}{`⟨*false*⟩`}` If any of the `\VerifyCommand` tests passed, do the `true` clause. If none of them passed, do the `false` clause.

```
\newcommand*{\IfVerifyCommandPassed}
    {%
        \ifbool{VERCMD@passed}%
            {\let\VERCMD@tempa\@firstoftwo}%
            {\let\VERCMD@tempa\@secondoftwo}%
        \VERCMD@tempa%
    }
```

`\IfVerifyCommandFailed` `{`⟨*true*⟩`}{`⟨*false*⟩`}` If any of the `\VerifyCommand` tests failed, do the `true` clause. If none of them failed, do the `false` clause.

```
\newcommand*{\IfVerifyCommandFailed}
    {%
        \ifbool{VERCMD@failed}%
            {\let\VERCMD@tempa\@firstoftwo}%
            {\let\VERCMD@tempa\@secondoftwo}%
        \VERCMD@tempa%
    }
```

`\IfVerifyCommand` `[`⟨*yourpackage*⟩`]` `[`⟨*theirpackage*⟩`]` `{`⟨`\commandname`⟩`}` `{`⟨MD5 *checksum*⟩`}` `{`⟨*true*⟩`}` `{`⟨*false*⟩`}`

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

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

Only if the package is enabled:

```
    \ifbool{VERCMDenable}{%
```

Default to an un-detected definition type:

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

Will become true if found `\NewDocumentCommand` or related.

```
    \boolfalse{VERCMD@this@ltxcmd}
```

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

```
%    % NewDocumentCommand:
     \ifcsdef{\cs_to_str:N #3~code}%
        {%
            \booltrue{VERCMD@this@ltxcmd}%
            \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 shown in place of the checksum.

```
                               {%
%                                   % newcommand:
                                    \ifdef{#3}%
```

```
                                        {\VERCMD@mdfive#3}%
                                        {}%
                                }%
                        }%
                }%
        }%
```

If the checksum matches the expected value, do the following `true` clause, else do the following `false` clause. Also track the true and false tests for `\IfVerifyCommandPassed` and `\IfVerifyCommandFailed`.

```
    \ifdefstring{\VERCMD@temp}{#4}%
        {%
            \booltrue{VERCMD@passed}%
            \let\VERCMD@tempa\@firstoftwo%
        }%
        {%
            \booltrue{VERCMD@failed}%
            \let\VERCMD@tempa\@secondoftwo%
        }%
    }% if package enabled
    {\let\VERCMD@tempa\@firstoftwo}% if package not enabled
    \VERCMD@tempa%
}
```

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

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

```
\NewDocumentCommand{\VerifyCommand}{O{} O{} m m}{%
    \ifblank{#1}{% #1 blank
        \IfVerifyCommand{#3}{#4}{}{\VERCMDWarning{}{}{#3}}%
    }% #1 blank
    {% #1 given
        \ifblank{#2}{% #2 blank
            \IfVerifyCommand[#1]{#3}{#4}{}{\VERCMDWarning{#1}{}{#3}}%
        }% #2 blank
        {% #2 given
            \IfVerifyCommand[#1][#2]{#3}{#4}{}{\VERCMDWarning{#1}{#2}{#3}}%
        }% #2 given
    }% #1 given
}
```

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

Test the begin section of the environment.

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

Only if the package is enabled:

```
    \ifbool{VERCMDenable}{%
```

Default to an un-detected definition type:

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

Will become true if found `\NewDocumentCommand` or related.

```
    \boolfalse{VERCMD@this@ltxcmd}
```

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

```
%    % NewDocumentEnvironment:
     \ifcsdef{environment~#3~code}%
        {%
            \booltrue{VERCMD@this@ltxcmd}%
            \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}%
                      {}%
                }%
        }%
```

Do the first or second next argument depending on a match:

```
    \ifdefstring{\VERCMD@temp}{#4}%
        {%
            \booltrue{VERCMD@passed}%
            \let\VERCMD@tempa\@firstoftwo%
        }%
        {%
            \booltrue{VERCMD@failed}%
            \let\VERCMD@tempa\@secondoftwo%
        }%
```

```
    }% if package enabled
    {\let\VERCMD@tempa\@firstoftwo}% if package not enabled
    \VERCMD@tempa%
}
```

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

Test the end section of the environment.

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

Only if the package is enabled:

```
    \ifbool{VERCMDenable}{%
```

Default to an un-detected definition type:

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

Will become true if found `\NewDocumentCommand` or related.

```
    \boolfalse{VERCMD@this@ltxcmd}
```

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

```
%    % end DocumentEnvironment:
     \ifcsdef{environment~#3~end~aux~}%
        {%
            \booltrue{VERCMD@this@ltxcmd}%
            \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%
                }%
                {}%
        }%
```

Do the first or second next argument depending on a match:

```
    \ifdefstring{\VERCMD@temp}{#4}%
        {%
            \booltrue{VERCMD@passed}%
            \let\VERCMD@tempa\@firstoftwo%
```

```
        }%
        {%
            \booltrue{VERCMD@failed}%
            \let\VERCMD@tempa\@secondoftwo%
        }%
    }% if package enabled
    {\let\VERCMD@tempa\@firstoftwo}% if package not enabled
    \VERCMD@tempa%
}
```

`\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}{%
%   begin:
    \ifblank{#1}{% #1 blank
        \IfVerifyEnvironmentBegin{#3}{#4}%
            {}%
            {\VERCMDWarning{}{}{#3}}%
    }% #1 blank
    {% #1 given
        \ifblank{#2}{% #2 blank
            \IfVerifyEnvironmentBegin[#1]{#3}{#4}%
                {}%
                {\VERCMDWarning{#1}{}{#3}}%
        }% #2 blank
        {% #2 given
            \IfVerifyEnvironmentBegin[#1][#2]{#3}{#4}%
                {}%
                {\VERCMDWarning{#1}{#2}{#3}}%
        }% #2 given
    }% #1 given
% end:
    \ifblank{#1}{% #1 blank
        \IfVerifyEnvironmentEnd{#3}{#5}%
            {}%
            {\VERCMDWarning{}{}{(end)#3}}%
    }% #1 blank
    {% #1 given
        \ifblank{#2}{% #2 blank
            \IfVerifyEnvironmentEnd[#1]{#3}{#5}%
                {}%
                {\VERCMDWarning{#1}{}{(end)#3}}%
        }% #2 blank
        {% #2 given
            \IfVerifyEnvironmentEnd[#1][#2]{#3}{#5}%
                {}%
                {\VERCMDWarning{#1}{#2}{(end)#3}}%
        }% #2 given
    }% #1 given
}
```

```
\ExplSyntaxOff
```

5.7 Verify infrastructure

The low-level infrastructure for `\NewDocumentCommand` and related may change on occasion, causing a change in the resulting code and a verification error for such code. The following is done while generating the documentation for the verifycommand package, and verifies the definitions of the underlying infrastructure and issues a warning if any has changed.

In the source for the following, `ltxcmd.dtx` uses `@@` which is replaced by `__cmd`.

(Testing silently occurs here, without adding text to the documentation.)

6 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

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.