The minted package: Highlighted source code in LaTeX

Geoffrey M. Poore
gpoore@gmail.com
github.com/gpoore/minted

Originally created and maintained (2009-2013) by
Konrad Rudolph

v2.0alpha from 2013/07/30

Abstract

minted is a package that facilitates expressive syntax highlighting using the powerful Pygments library. The package also provides options to customize the highlighted source code output.

Current status

minted was created in 2009 by Konrad Rudolph. Geoffrey Poore agreed to take over minted maintenance in March of 2013, since his PythonTeX package also provides an interface to Pygments.

minted is currently in an alpha release for v2.0. The goal is to close all current issues, including those on the old Google Code site, before the full v2.0 release. During this time of transition, users who need maximum stability are encouraged to use minted 1.7 or PythonTeX. The release on CTAN will only be updated once v2.0 stabilizes.

License

Contents

1 Introduction

minted is a package that allows formatting source code in LaTeX. For example:

```
\begin{minted}{language}
  code
\end{minted}
```

will highlight a piece of code in a chosen language. The display can be customized by a number of arguments and colour schemes.

Unlike some other packages, most notably listings, minted requires the installation of additional software, Pygments. This may seem like a disadvantage, but there are also significant advantages.

Pygments provides superior syntax highlighting compared to conventional packages. For example, listings basically only highlights strings, comments and keywords. Pygments, on the other hand, can be completely customized to highlight any kind of token the source language might support. This might include special formatting sequences inside strings, numbers, different kinds of identifiers and exotic constructs such as HTML tags.

Some languages make this especially desirable. Consider the following Ruby code as an extreme, but at the same time typical, example:

for compatibility with earlier versions

```ruby
class Foo
  def init
    pi = Math::PI
    @var = "Pi is approx. #{pi}"
  end
end
```

Here we have four different colors for identifiers (five, if you count keywords) and escapes from inside strings, none of which pose a problem for Pygments.

Additionally, installing Pygments is actually incredibly easy (see the next section).

2 Installation

2.1 Prerequisites

Pygments is written in Python, so make sure that you have at Python 2.6 or later installed on your system. This may be easily checked from the command line:

```
$ python --version
Python 2.7.5
```

If you don't have Python installed, you can download it from the Python website or use your operating system's package manager.

Some Python distributions include Pygments (see some of the options under "Alternative Implementations" on the Python site). Otherwise, you will need to install Pygments manually.

This may be done by installing setuptools, which facilitates the distribution of Python applications. You can then install Pygments using the following command:

```
$ sudo easy_install Pygments
```

Under Windows, you will not need the `sudo`, but may need to run the command prompt as administrator.

If you already have Pygments installed, be aware that minted requires at least version 1.2. Since support for new languages and new features are added with each release, it would probably be good to use version 1.5 or later.

2.2 Required packages

minted requires that the following packages be available and reasonably up to date on your system. All of these ship with recent TEX distributions.

- `keyval`
- `fancyvrb`
- `float`
- `ifthen`
- `calc`
- `ifplatform`
- `pdftexcmds`
- `etoolbox`
- `xstring`
- `xcolor`

2.3 Installing minted

You can probably install minted with your TEX distributions package manager. Otherwise, you can install it manually by following the directions below.

If the file `minted.sty` doesn't exist yet, we first have to create it. If you're using a system that supports the make command, then you can simply type the following command in the folder where you've extracted the minted package code:

```
$ make
```

Alternatively, you may download this file separately from the project's homepage, or create it manually by executing the command

```
$ tex minted.ins
```

on the command line.

Finally, we have to install the file so that TEX is able to find it. In order to do that, please refer to the TEX FAQ. If you just want to experiment with the latest version, you could locate your current `minted.sty` in your TEX installation and replace it with the latest version. Or you could just put the latest `minted.sty` in the same directory as the file you wish to use it with.

3 Basic usage

3.1 Preliminary

Since minted makes calls to the outside world (that is, Pygments), you need to tell the LATEX processor about this by passing it the `-shell-escape` option or it won't allow such calls. In effect, instead of calling the processor like this:

```
$ latex input
```

you need to call it like this:

```
$ latex -shell-escape input
```

The same holds for other processors, such as `pdflatex` or `xelatex`.

You should be aware that using `-shell-escape` allows LATEX to run potentially arbitrary commands on your system. It is probably best to use `-shell-escape` only when you need it, and to use only it with documents from trusted sources.

3.2 A minimal complete example

The following file `minimal.tex` shows the basic usage of minted.

```
\documentclass{article}

\usepackage{minted}

\begin{document}
```

```
\begin{minted}{c}
int main() {
    printf("hello, world");
    return 0;
}
\end{minted}
\end{document}
```

By compiling the source file like this:

```
$ pdflatex -shell-escape minimal
```

we end up with the following output in `minimal.pdf`:

```c
int main() {
    printf("hello, world");
    return 0;
}
```

3.3 Formatting source code

minted

Using minted is straightforward. For example, to highlight some Python source code we might use the following code snippet (result on the right):

```
\begin{minted}{python}
def boring(args = None):
    pass
\end{minted}
```

```python
def boring(args = None):
    pass
```

Optionally, the environment accepts a number of options in `key=value` notation, which are described in more detail below.

\mint

For a single line of source code, you can alternatively use a shorthand notation:

```
\mint{python}|import this|
```

```
import this
```

The code is delimited by a pair of identical characters, similar to how `\verb` works. The complete syntax is `\mint[`⟨*options*⟩`]{`⟨*language*⟩`}`⟨*delim*⟩⟨*code*⟩⟨*delim*⟩, where the code delimiter can be almost any punctuation character. Again, this command supports a number of options described below.

Note that the `\mint` command **is not for inline use**. Rather, it is a shortcut for `minted` when only a single line of code is present. The `\mintinline` command is provided for inline use.

\mintinline

Code can be typeset inline:

```
X\mintinline{python}{print(x**2)}X
```

X`print(x**2)`X

The syntax is `\mintinline[`⟨*options*⟩`]{`⟨*language*⟩`}`⟨*delim*⟩⟨*code*⟩⟨*delim*⟩. The delimiters can be a pair of characters, as for `\mint`. They can also be a matched pair of curly braces, `{}`.

The command has been carefully crafted so that in most cases it will function correctly when used inside other commands.[1]

`\inputminted` Finally, there's the `\inputminted` command to read and format whole files. Its syntax is `\inputminted[`⟨*options*⟩`]{`⟨*language*⟩`}{`⟨*filename*⟩`}`.

3.4 Using different styles

`\usemintedstyle` Instead of using the default style you may choose another stylesheet provided by Pygments. This may be done via the following:

`\usemintedstyle``{name}`

The full syntax is `\usemintedstyle[`⟨*language*⟩`]{`⟨*style*⟩`}`. The style may be set for the document as a whole (no language specified), or only for a particular language. Note that the style may also be set via `\setminted` and via the optional argument for each command and environment.[2]

To get a list of all available stylesheets, see the online demo at the Pygments website or execute the following command on the command line:

```
$ pygmentize -L styles
```

Creating your own styles is also easy. Just follow the instructions provided on the website.

3.5 Supported languages

Pygments supports over 150 different programming languages, template languages, and other markup languages. To see an exhaustive list of the currently supported languages, use the command

```
$ pygmentize -L lexers
```

[1] For example, **`\mintinline`** works in footnotes! The main exception is when the code contains the percent `%` and hash `#` characters.

[2] Version 2.0 added the optional language argument and removed the restriction that the command be used in the preamble.

4 Floating listings

`listing` minted provides the `listing` environment to wrap around a source code block. This puts the code into a floating box. You can also provide a `\caption` and a `\label` for such a listing in the usual way (that is, as for the `table` and `figure` environments):

```
\begin{listing}[H]
  \mint{cl}/(car (cons 1 '(2)))/
  \caption{Example of a listing.}
  \label{lst:example}
\end{listing}

Listing \ref{lst:example} contains an example of a listing.
```

will yield:

```
(car (cons 1 '(2)))
```

Listing 1: Example of a listing.

Listing 1 contains an example of a listing.

`\listoflistings` The `\listoflistings` macro will insert a list of all (floated) listings in the document:

```
\listoflistings
```

List of listings

1 Example of a listing. 9

`\listingscaption` The string "Listing" in a listing's caption can be changed. To do this, simply redefine the macro `\listingscaption`, for example:

```
\renewcommand{\listingscaption}{Program code}
```

`\listoflistingscaption` Likewise, the caption of the listings list, "List of listings," can be changed by redefining `\listoflistingscaption`:

```
\renewcommand{\listoflistingscaption}{List of program codes}
```

5 Options

5.1 Package options

section
chapter

To control how LaTeX counts the `listing` floats, you can pass either the `section` or `chapter` option when loading the minted package. For example, the following will cause listings to be counted by section:

```
\usepackage[section]{minted}
```

cache

minted works by saving code to a temporary file, highlighting the code via Pygments and saving the output to another temporary file, and inputting the output into the LaTeX document. This process can become quite slow if there are many chunks of code to highlight. To avoid this, the package provides a `cache` option.

The `cache` option creates a directory `.minted-`⟨*jobname*⟩ in the document's root directory. Files of highlighted code are stored in this directory, so that the code will not have to be highlighted again in the future. In most cases, caching will significantly speed up document compilation.

Cached files that are no longer in use are automatically deleted.[3]

langlinenos

minted uses the fancyvrb package behind the scenes for the code typesetting. fancyvrb provides an option `firstnumber` that allows the starting line number of an environment to be specified. For convenience, there is an option `firstnumber=last` that allows line numbering to pick up where it left off. The `langlinenos` option makes `firstnumber` work for each language individually with all `minted` and `\mint` usages. For example, consider the code and output below.

```
\begin{minted}[linenos]{python}
def f(x):
    return x**2
\end{minted}

\begin{minted}[linenos]{ruby}
def func
    puts "message"
end
\end{minted}

\begin{minted}[linenos, firstnumber=last]{python}
def g(x):
    return 2*x
\end{minted}
```

[3] This depends on the main auxiliary file not being deleted or becoming corrupted. If that happens, you could simply delete the cache directory and start over.

```
def f(x):
    return x**2

def func
    puts "message"
end

def g(x):
    return 2*x
```

Without the `langlinenos` option, the line numbering in the second Python environment would not pick up where the first Python environment left off.

5.2 Macro option usage

All minted highlighting commands accept the same set of options. Options are specified as a comma-separated list of `key=value` pairs. For example, we can specify that the lines should be numbered:

```
\begin{minted}[linenos=true]{c++}
#include <iostream>
int main() {
    std::cout << "Hello "
              << "world"
              << std::endl;
}
\end{minted}
```

```
#include <iostream>
int main() {
    std::cout << "Hello "
              << "world"
              << std::endl;
}
```

An option value of `true` may also be omitted entirely (including the "="). To customize the display of the line numbers further, override the `\theFancyVerbLine` command. Consult the fancyvrb documentation for details.

`\mint` accepts the same options:

```
\mint[linenos]{perl}|$x=~/foo/|
```

```
$x=~/foo/
```

Here's another example: we want to use the LaTeX math mode inside comments:

```
\begin{minted}[mathescape]{python}
# Returns $\sum_{i=1}^{n}i$
def sum_from_one_to(n):
    r = range(1, n + 1)
    return sum(r)
\end{minted}
```

```
# Returns $\sum_{i=1}^{n} i$
def sum_from_one_to(n):
    r = range(1, n + 1)
    return sum(r)
```

To make your LaTeX code more readable you might want to indent the code inside a `minted` environment. The option `gobble` removes these unnecessary whitespace characters from the output:

```
\begin{minted}[gobble=2,
  showspaces]{python}
  def boring(args = None):
      pass
\end{minted}

versus

\begin{minted}[showspaces]{python}
  def boring(args = None):
      pass
\end{minted}
```

```
def␣boring(args␣=␣None):
␣␣␣␣pass

versus

␣␣def␣boring(args␣=␣None):
␣␣␣␣␣␣pass
```

\setminted You may wish to set options for the document as a whole, or for an entire language. This is possible via `\setminted`. The syntax is `\setminted[`⟨*language*⟩`]{`⟨*key=value,...*⟩`}`. Language-specific options override document-wide options, which in turn are overridden by individual command and environment options.

5.3 Available options

Following is a full list of available options. For more detailed option descriptions please refer to the fancyvrb and Pygments documentation.

baselinestretch (`auto`|dimension) (default: `auto`)
Value to use as for baselinestretch inside the listing.

bgcolor (string) (default: *none*)
Background color of the listing. Notice that the value of this option must *not* be a color command. Instead, it must be a color *name*, given as a string, of a previously-defined color:

```
\definecolor{bg}{rgb}{0.95,0.95,0.95}
\begin{minted}[bgcolor=bg]{php}
<?php
  echo "Hello, $x";
?>
\end{minted}
```

```
<?php
  echo "Hello, $x";
?>
```

Unlike the other options, this option is currently only supported for individual commands and environments; there is not support at the language and document-wide levels.

codetagify (list of strings) (default: highlight `XXX`, `TODO`, `BUG`, and `NOTE`)
Highlight special code tags in comments and docstrings.

encoding (string) (default: system-specific)
Sets the file encoding that Pygments expects.

outencoding (string) (default: system-specific)
Sets the file encoding that Pygments uses for highlighted output. Overrides any encoding previously set via `encoding`.

`firstline` (integer) (default: `1`)
The first line to be shown. All lines before that line are ignored and do not appear in the output.

`firstnumber` (`auto`|integer) (default: `auto` = 1)
Line number of the first line.

`fontfamily` (family name) (default: `tt`)
The font family to use. `tt`, `courier` and `helvetica` are pre-defined.

`fontseries` (series name) (default: `auto` – the same as the current font)
The font series to use.

`fontsize` (font size) (default: `auto` – the same as the current font)
The size of the font to use, as a size command, e.g. `\footnotesize`.

`fontshape` (font shape) (default: `auto` – the same as the current font)
The font shape to use.

`formatcom` (command) (default: *none*)
A format to execute before printing verbatim text.

`frame` (`none`|`leftline`|`topline`|`bottomline`|`lines`|`single`) (default: `none`)
The type of frame to put around the source code listing.

`framerule` (dimension) (default: `0.4pt`)
Width of the frame.

`framesep` (dimension) (default: `\fboxsep`)
Distance between frame and content.

`funcnamehighlighting` (boolean) (default: `true`)
[For PHP only] If `true`, highlights built-in function names.

`gobble` (integer) (default: `0`)
Remove the first n characters from each input line.

`keywordcase` (string) (default: `'lower'`)
Changes capitalization of keywords. Takes `'lower'`, `'upper'`, or `'capitalize'`.

`label` ([string]string) (default: *empty*)
Add a label to the top, the bottom or both of the frames around the code. See the fancyvrb documentation for more information and examples. *Note:* This does *not* add a `\label` to the current listing. To achieve that, use a floating environment (section 4) instead.

`labelposition` (`none`|`topline`|`bottomline`|`all`) (default: `topline`, `all` or *none*)
Position where to print the label (see above; default: `topline` if one label is defined, `all` if two are defined, *none* else). See the fancyvrb documentation for more information.

`lastline` (integer) (default: *last line of input*)
The last line to be shown.

`linenos` (boolean) (default: `false`)
Enables line numbers. In order to customize the display style of line numbers, you need to redefine the `\theFancyVerbLine` macro:

```
\renewcommand{\theFancyVerbLine}{\sffamily
  \textcolor[rgb]{0.5,0.5,1.0}{\scriptsize
  \oldstylenums{\arabic{FancyVerbLine}}}}
\begin{minted}[linenos,
  firstnumber=11]{python}
def all(iterable):
    for i in iterable:
        if not i:
            return False
    return True
\end{minted}
```

```
11 def all(iterable):
12     for i in iterable:
13         if not i:
14             return False
15     return True
```

`numbers` (`left`|`right`) (default: *none*)
Essentially the same as `linenos`, except the side on which the numbers appear may be specified.

`mathescape` (boolean) (default: `false`)
Enable LaTeX math mode inside comments. Do *not* use spaces inside math mode—they will be rendered like other full-width verbatim spaces. Usage as in package listings.

`numberblanklines` (boolean) (default: `true`)
Enables or disables numbering of blank lines.

`numbersep` (dimension) (default: `12pt`)
Gap between numbers and start of line.

`obeytabs` (boolean) (default: `false`)
Treat tabs as tabs instead of converting them to spaces.

`python3` (boolean) (default: `false`)
[For PythonConsoleLexer only] Specifies whether Python 3 highlighting is applied.

`resetmargins` (boolean) (default: `false`)
Resets the left margin inside other environments.

`rulecolor` (color command) (default: *black*)
The color of the frame.

`samepage` (boolean) (default: `false`)
Forces the whole listing to appear on the same page, even if it doesn't fit.

`showspaces` (boolean) (default: `false`)
Enables visible spaces: `visible␣spaces`.

`showtabs` (boolean) (default: `false`)
Enables visible tabs—only works in combination with `obeytabs`.

`startinline` (boolean) (default: `false`)

[For PHP only] Specifies that the code starts in PHP mode, i.e. leading `<?php` is omitted.

`style` (string) (default: *default*)
Sets the stylesheet used by Pygments.

`stepnumber` (integer) (default: `1`)
Interval at which line numbers appear.

`tabsize` (integer) (default: `8`)
The number of spaces a tab is equivalent to if `obeytabs` is not active.

`texcl` (boolean) (default: `false`)
Enables LaTeX code inside comments. Usage as in package listings.

`texcomments` (boolean) (default: `false`)
Enables LaTeX code inside comments. The newer name for `texcl`.

`xleftmargin` (dimension) (default: `0`)
Indentation to add before the listing.

`xrightmargin` (dimension) (default: `0`)
Indentation to add after the listing.

6 Defining shortcuts

Large documents with a lot of listings will nonetheless use the same source language and the same set of options for most listings. Always specifying all options is redundant, a lot to type and makes performing changes hard.

One option is to use `\setminted`, but even then you must still specify the language each time.

minted therefore defines a set of commands that lets you define shortcuts for the highlighting commands. Each shortcut is specific for one programming language.

`\newminted` `\newminted` defines a new alias for the `minted` environment:

```
\newminted{cpp}{gobble=2,linenos}

\begin{cppcode}
  template <typename T>
  T id(T value) {
      return value;
  }
\end{cppcode}
```

```
1 template <typename T>
2 T id(T value) {
3     return value;
4 }
```

If you want to provide extra options on the fly, or override existing default options, you can do that, too:

```
\newminted{cpp}{gobble=2,linenos}

\begin{cppcode*}{linenos=false,
                 frame=single}
  int const answer = 42;
\end{cppcode*}
```

```
int const answer = 42;
```

Notice the star "⋆" behind the environment name—due to restrictions in fancyvrb's handling of options, it is necessary to provide a *separate* environment that accepts options, and the options are *not* optional on the starred version of the environment.

The default name of the environment is ⟨*language*⟩`code`. If this name clashes with another environment or if you want to choose an own name for another reason, you may do so by specifying it as the first argument: `\newminted[`⟨*environment name*⟩`]{`⟨*language*⟩`}{`⟨*options*⟩`}`.

`\newmint` The above macro only defines shortcuts for the `minted` environment. The main reason is that the short command form `\mint` often needs different options—at the very least, it will generally not use the `gobble` option. A shortcut for `\mint` is defined using `\newmint[`⟨*macro name*⟩`]{`⟨*language*⟩`}{`⟨*options*⟩`}`. The arguments and usage are identical to `\newminted`. If no ⟨*macro name*⟩ is specified, ⟨*language*⟩ is used.

```
\newmint{perl}{showspaces}

\perl/my $foo = $bar;/
```

```
my␣$foo␣=␣$bar;
```

`\newmintinline` This creates custom versions of `\mintinline`. The syntax is the same as that for `\newmint`: `\newmintinline[`⟨*macro name*⟩`]{`⟨*language*⟩`}{`⟨*options*⟩`}`. If a ⟨*macro name*⟩ is not specified, then the created macro is called `\`⟨*language*⟩`inline`.

```
\newmintinline{perl}{showspaces}

X\perlinline/my $foo = $bar;/X
```

```
Xmy␣$foo␣=␣$bar;X
```

`\newmintedfile` This creates custom versions of `\inputminted`. The syntax is

`\newmintedfile[`⟨*macro name*⟩`]{`⟨*language*⟩`}{`⟨*options*⟩`}`

If no ⟨*macro name*⟩ is given, then the macro is called `\`⟨*language*⟩`file`.

7 Troubleshooting

In some cases, minted may not give the desired result due to other document settings that it cannot control. Common issues are described below, with workarounds or solutions. You may also wish to search tex.stackexchange.com or ask a question there, if you are working with minted in a non-typical context.

- **Tilde characters ~ are raised, almost like superscripts.** This is a font issue. You need a different font encoding, possibly with a different font. Try `\usepackage[T1]{fontenc}` plus `\usepackage{lmodern}`, or something similar.
- **Extended characters do not work inside minted commands and environments, even when the inputenc package is used.** Version 2.0 adds support for extended characters under the pdfTeX engine. But if you need characters that are not supported by inputenc, you should use the XeTeX or LuaTeX engines instead.
- **The caption package produces an error when `\captionof` and other commands are used in combination with minted.** Load the caption package with the option `compatibility=false`.

Acknowledgements

Konrad Rudolph: Special thanks to Philipp Stephani and the rest of the guys from comp.text.tex and tex.stackexchange.com.

Version History

2.0alpha (2013/07/30)

- Added the package option `cache`. This significantly increases compilation speed by caching old output. For example, compiling the documentation is around 5x faster.
- New inline command `\mintinline`. Custom versions can be created via `\newmintinline`. The command works inside other commands (for example, footnotes) in most situations, so long as the percent and hash characters are avoided.
- The new `\setminted` command allows options to be specified at the document and language levels.
- All extended characters (Unicode, etc.) supported by `inputenc` now work under the pdfTeX engine. This involved using `\detokenize` on everything prior to saving.
- New package option `langlinenos` allows line numbering to pick up where it left off for a given language when `firstnumber=last`.
- New options, including `style`, `encoding`, `outencoding`, `codetagify`, `keywordcase`, `texcomments` (same as `texcl`), `python3` (for the `PythonConsoleLexer`), and `numbers`.

- `\usemintedstyle` now takes an optional argument to specify the style for a particular language, and works anywhere in the document.
- `xcolor` is only loaded if `color` isn't, preventing potential package clashes.

8 Implementation

8.1 Required packages

Load required packages. For compatibility reasons, most old functionality should be supported with the original set of packages. More recently added packages, such as `etoolbox` and `xstring`, should only be used for new features when possible.

```
\RequirePackage{keyval}
\RequirePackage{fancyvrb}
\RequirePackage{float}
\RequirePackage{ifthen}
\RequirePackage{calc}
\RequirePackage{ifplatform}
\RequirePackage{pdftexcmds}
\RequirePackage{etoolbox}
\RequirePackage{xstring}
```

Make sure that either `color` or `xcolor` is loaded.

```
\AtBeginDocument{\@ifpackageloaded{color}{}{\RequirePackage{xcolor}}}
```

8.2 Package options

`\minted@float@within` Define an option that controls the section numbering of the `listing` float.

```
\DeclareOption{chapter}{\def\minted@float@within{chapter}}
\DeclareOption{section}{\def\minted@float@within{section}}
```

`minted@cache` Define an option that determines whether highlighted content is cached. We use a boolean to keep track of its state.

```
\newboolean{minted@cache}
\DeclareOption{cache}{%
  \minted@cachetrue
  \AtEndOfPackage{\ProvideDirectory{\minted@cachedir}}%
}
```

`langlinenos` Define an option that makes all `minted` environments and `\mint` commands for a given language share cumulative line numbering (if `firstnumber=last`).

```
\newboolean{minted@langlinenos}
\DeclareOption{langlinenos}{\minted@langlinenostrue}
```

Process package options.

```
\ProcessOptions\relax
```

8.3 Caching and temp files

`\minted@infile` Define a default name for files of highlighted content that are brought it. Caching will redefine this. We start out with the default, non-caching value.

```
\newcommand{\minted@infile}{\jobname.out.pyg}
```

`\minted@cachedir` Define a macro with the name of the cache directory. This uses a `.minted-` prefix followed by a sanitized `\jobname` (spaces and asterisks replaced).

```
\StrSubstitute{\jobname}{ }{_}[\minted@jobname]
\StrSubstitute{\minted@jobname}{"}{}[\minted@jobname]
\StrSubstitute{\minted@jobname}{*}{-}[\minted@jobname]
\newcommand{\minted@cachedir}{.minted-\minted@jobname}
```

`minted-xetex-hasher.py` XeTeX doesn't have built-in hashing capabilities, so we create a temporary script to handle that. (pdfTeX has `\pdfmdfivesum`, and LuaTeX has equivalent capabilities through Lua; a common hashing interface for both engines is provided by the pdftexcmds package.)

```
\expandafter\ifx\csname XeTeXinterchartoks\endcsname\relax
\else
\begin{VerbatimOut}[commandchars=!\[\]]{minted-xetex-hasher.py}
import sys
import hashlib
hasher = hashlib.sha1()
f = open('!jobname.mintedcmd', 'rb')
hasher.update(f.read())
f.close()
f = open(sys.argv[1], 'rb')
hasher.update(f.read())
f.close()
f = open('!jobname.mintedmd5', 'w')
macro = r'\edef\minted@hash{' + hasher.hexdigest() + '}%\n'
f.write('\\makeatletter\n')
f.write(macro)
f.write('\\makeatother\n')
f.close()
\end{VerbatimOut}
\fi
```

We need a way to track the cache files that are created, and delete those that are not in use.

`\minted@cachefiles` This is a list of the current cache files.

```
\newcommand{\minted@cachefiles}{}
```

`\minted@addcachefile` This adds a file to the list of cache files. It also creates a macro involving the hash, so that the current usage of the hash can be easily checked.

```
\newcommand{\minted@addcachefile}[1]{%
  \expandafter\gdef\expandafter\minted@cachefiles\expandafter{%
    \minted@cachefiles,#1}%
  \expandafter\gdef\csname minted@current@#1\endcsname{}%
}
```

`\minted@savecachefiles` We need to be able to save the list of cache files to the `.aux` file, so that we can reload it on the next run.

```
\newcommand{\minted@savecachefiles}{%
  \immediate\write\@mainaux{%
    \string\gdef\string\minted@oldcachefiles\string{%
      \minted@cachefiles\string}}%
}
```

`\minted@cleancache` Clean up old cache files that are no longer in use.

```
\newcommand{\minted@cleancache}{%
  \ifthenelse{\boolean{minted@cache}}{%
    \ifcsname minted@oldcachefiles\endcsname
      \renewcommand{\do}[1]{%
        \ifthenelse{\equal{##1}{}}{}{%
          \ifcsname minted@current@##1\endcsname\else
            \DeleteFile[\minted@cachedir]{##1.pygtex}%
          \fi
        }%
      }%
      \expandafter\docsvlist\expandafter{\minted@oldcachefiles}%
    \else
    \fi
  }{}%
}
```

At the end of the document, save the list of cache files and clean the cache.

```
\ifthenelse{\boolean{minted@cache}}%
  {\AtEndDocument{%
    \minted@savecachefiles
    \minted@cleancache}}%
  {}%
```

8.4 OS interaction

We need system-dependent macros for communicating with the "outside world."

`\DeleteFile` Delete a file. Define conditionally in case an equivalent macro has already been defined.

```
\ifwindows
  \providecommand{\DeleteFile}[2][]{%
    \ifthenelse{\equal{#1}{}}{\immediate\write18{del "#2"}}%
      {\immediate\write18{del "#1\@backslashchar #2"}}}
\else
  \providecommand{\DeleteFile}[2]{%
    \ifthenelse{\equal{#1}{}}{\immediate\write18{rm "#2"}}%
      {\immediate\write18{rm "#1/#2"}}}
\fi
```

`\ProvideDirectory` We need to be able to create a directory, if it doesn't already exist. This is primarily for storing cached highlighted content.

```
\ifwindows
  \newcommand{\ProvideDirectory}[1]{%
    \immediate\write18{if not exist "#1" mkdir "#1"}}
\else
  \newcommand{\ProvideDirectory}[1]{%
    \immediate\write18{mkdir -p "#1"}}
\fi
```

`\TestAppExists` Determine whether a given application exists.

Usage is a bit roundabout, but has been retained for backward compatibility. To test whether an application exists, use the following code:

```
\TestAppExists{appname}
\ifthenelse{\boolean{AppExists}}{app exists}{app doesn't exist}
```

```
\newboolean{AppExists}
\newread\minted@appexistsfile
\newcommand{\TestAppExists}[1]{
  \ifwindows
```

On Windows, we need to use path expansion and write the result to a file. If the application doesn't exist, the file will be empty (except for a newline); otherwise, it will contain the full path of the application.

```
    \DeleteFile{\jobname.aex}
    \immediate\write18{for \string^\@percentchar i in (#1.exe #1.bat #1.cmd)
      do set >\jobname.aex <nul: /p
```

```
      x=\string^\@percentchar \string~$PATH:i>>\jobname.aex}
    %$ <- balance syntax highlighting
    \immediate\openin\minted@appexistsfile\jobname.aex
    \expandafter\def\expandafter\@tmp@cr\expandafter{\the\endlinechar}
    \endlinechar=-1\relax
    \readline\minted@appexistsfile to \minted@apppathifexists
    \endlinechar=\@tmp@cr
    \ifthenelse{\equal{\minted@apppathifexists}{}}
     {\AppExistsfalse}
     {\AppExiststrue}
    \immediate\closein\minted@appexistsfile
    \DeleteFile{\jobname.aex}
    \immediate\typeout{file deleted}
  \else
```

On Unix-like systems, we do a straightforward `which` test and create a file upon success, whose existence we can then check.

```
    \immediate\write18{which #1 && touch \jobname.aex}
    \IfFileExists{\jobname.aex}
      {\AppExiststrue
        \DeleteFile{\jobname.aex}}
      {\AppExistsfalse}
  \fi
}
```

8.5 Option processing

We need macros for storing options that will later be passed to the command line. These are defined at the global (`g`), language (`lang`), and command or environment (`cmd`) levels, so that settings can be specified at various levels of hierarchy. The language macro is actually a placeholder. Each individual language will create a `\minted@optlang`⟨*language*⟩ macro. The current language will be tracked using `\minted@lang`.

`\minted@optg`

```
\newcommand{\minted@optg}{}
```

`\minted@lang`

```
\let\minted@lang\@empty
```

`\minted@optlang`

```
\newcommand{\minted@optlang}{}
```

\minted@optcmd

```
\newcommand{\minted@optcmd}{}
```

\minted@checklang We need a way to check whether a language has had all its option macros created.

```
\newcommand{\minted@checklang}{%
  \ifcsname minted@optlang\minted@lang\endcsname\else
    \expandafter\gdef\csname minted@optlang\minted@lang\endcsname{}%
  \fi
  \ifcsname minted@optlang\minted@lang @extra\endcsname\else
    \expandafter\gdef\csname minted@optlang\minted@lang @extra\endcsname{}%
  \fi
}
```

\minted@resetoptcmd We need a macro that will reset command-level options to null values. This macro will be redefined as command-level options are specified, in such a way that the next time it is used it will reset all options to null values. An equivalent at the global and command levels is not provided, since those options should persist.

It is convenient always to reset the `extra` options, given the number of `extra` options and the proportional inconvenience of always having to check whether they have been added to the reset list.

```
\newcommand{\minted@resetoptcmd}{\@namedef{minted@optcmd@extra}{}}
```

We need a macro that will retrieve detokenized option values suitable for `\write18`. We create three versions, one for each level of options.

\minted@getoptg

```
\newcommand{\minted@getoptg}[1]{%
  \expandafter\detokenize%
  \expandafter\expandafter\expandafter{\csname minted@optg@#1\endcsname}}
```

\minted@getoptlang

```
\newcommand{\minted@getoptlang}[1]{%
  \expandafter\detokenize\expandafter\expandafter\expandafter{%
    \csname minted@optlang\minted@lang @#1\endcsname}}
```

\minted@getoptcmd

```
\newcommand{\minted@getoptcmd}[1]{%
  \expandafter\detokenize%
  \expandafter\expandafter\expandafter{\csname minted@optcmd@#1\endcsname}}
```

We need a macro that will register options as having been used (add the options that are in use to the list macro). As before, we create three versions, one per level.

\minted@regoptg

```
\newcommand{\minted@regoptg}[1]{%
  \ifcsname minted@optg@#1@reg\endcsname\else
    \expandafter\global\expandafter%
      \let\csname minted@optg@#1@reg\endcsname\@empty
    \expandafter\gdef\expandafter\minted@optg\expandafter{%
      \minted@optg\space\minted@getoptg{#1}}%
  \fi
}
```

\minted@regoptlang

```
\newcommand{\minted@regoptlang}[1]{%
  \ifcsname minted@optlang\minted@lang @#1@reg\endcsname\else
    \ifcsname minted@optlang\minted@lang\endcsname\else
      \expandafter\gdef\csname minted@optlang\minted@lang\endcsname{}%
    \fi
    \expandafter\global\expandafter\let%
      \csname minted@optlang\minted@lang @#1@reg\endcsname\@empty
    \expandafter\let\expandafter\minted@optlang%
      \csname minted@optlang\minted@lang\endcsname
    \expandafter\def\expandafter\minted@optlang\expandafter{%
      \minted@optlang\space\minted@getoptlang{#1}}%
    \expandafter\global\expandafter\let%
      \csname minted@optlang\minted@lang\endcsname\minted@optlang
    \let\minted@optlang\@empty
  \fi
}
```

\minted@regoptcmd

```
\newcommand{\minted@regoptcmd}[1]{%
  \ifcsname minted@optcmd@#1@reg\endcsname\else
    \expandafter\global\expandafter%
      \let\csname minted@optcmd@#1@reg\endcsname\@empty
    \expandafter\gdef\expandafter\minted@optcmd\expandafter{%
      \minted@optcmd\space\minted@getoptcmd{#1}}%
    \expandafter\gdef\expandafter\minted@resetoptcmd\expandafter{%
        \minted@resetoptcmd
        \@namedef{minted@optcmd@#1}{}}
  \fi
}
```

\minted@define@opt

Define a generic option with an optional default argument. If a `key` option is specified without `=value`, the default is assumed. Options are automatically created at all levels.

```
\newcommand{\minted@define@opt}[3][]{%
```

```
  \ifthenelse{\equal{#1}{}}%
    {\define@key{minted@optg}{#2}{\@namedef{minted@optg@#2}{#3}%
        \minted@regoptg{#2}}%
      \define@key{minted@optlang}{#2}{%
        \@namedef{minted@optlang\minted@lang @#2}{#3}%
        \minted@regoptlang{#2}}%
      \define@key{minted@optcmd}{#2}{\@namedef{minted@optcmd@#2}{#3}%
        \minted@regoptcmd{#2}}}%
    {\define@key{minted@optg}{#2}[#1]{\@namedef{minted@optg@#2}{#3}%
        \minted@regoptg{#2}}%
      \define@key{minted@optlang}{#2}[#1]{%
        \@namedef{minted@optlang\minted@lang @#2}{#3}%
        \minted@regoptlang{#2}}%
      \define@key{minted@optcmd}{#2}[#1]{\@namedef{minted@optcmd@#2}{#3}%
        \minted@regoptcmd{#2}}}%
}
```

\minted@define@optstyle Define an option for styles. These are defined independently, because styles need to be registered, so that the macros for a given style are only created and inputted once.

```
\newcommand{\minted@define@optstyle}{%
  \define@key{minted@optg}{style}{%
    \@namedef{minted@optg@style}{-P style=##1 -P commandprefix=PYG##1}%
    \minted@regoptg{style}\minted@regstyle{##1}}%
  \define@key{minted@optlang}{style}{%
    \@namedef{minted@optlang\minted@lang @style}%
      {-P style=##1 -P commandprefix=PYG##1}%
    \minted@regoptlang{style}\minted@regstyle{##1}}%
  \define@key{minted@optcmd}{style}{%
    \@namedef{minted@optcmd@style}{-P style=##1 -P commandprefix=PYG##1}%
    \minted@regoptcmd{style}\minted@regstyle{##1}}%
}
```

\minted@regstyle Register any used styles, and make sure that the definitions are available.

It's important that registration be done with `\def`. The style macros are `def`ed, so they will be local if included within a group. We need to make sure that we don't make the mistake of registering a style globally that is actually only available in a group.

We have to do some tricks with `\endlinechar` to prevent `\input` from inserting unwanted whitespace. That is primarily for inline commands, where it would introduce a line break.

```
\newcommand{\minted@regstyle}[1]{%
  \ifcsname minted@stylereg@#1\endcsname\else
    \expandafter\let\csname minted@stylereg@#1\endcsname\@empty
    \ifthenelse{\boolean{minted@cache}}%
```

```
      {\IfFileExists{\minted@cachedir/#1.pygstyle}{}{%
        \ifwindows
          \immediate\write18{pygmentize -S #1 -f latex
            -P commandprefix=PYG#1
            > \minted@cachedir\@backslashchar#1.pygstyle}%
        \else
          \immediate\write18{pygmentize -S #1 -f latex
            -P commandprefix=PYG#1
            > \minted@cachedir/#1.pygstyle}%
        \fi
        }%
        \expandafter\def\expandafter\@tmp@cr\expandafter{\the\endlinechar}%
        \endlinechar=-1\relax
        \input{\minted@cachedir/#1.pygstyle}%
        \endlinechar=\@tmp@cr}%
      {\immediate\write18{pygmentize -S #1 -f latex
        -P commandprefix=PYG#1 > \jobname.pyg}%
        \expandafter\def\expandafter\@tmp@cr\expandafter{\the\endlinechar}%
        \endlinechar=-1\relax
        \input{\jobname.pyg}%
        \endlinechar=\@tmp@cr}%
  \fi
}
```

\minted@define@switch Define a switch or boolean option, which is `true` when no value is specified.

```
\newcommand{\minted@define@switch}[3][]{
  \define@booleankey{minted@optg}{#2}
    {\@namedef{minted@optg@#2}{#3}\minted@regoptg{#2}}
    {\@namedef{minted@optg@#2}{#1}\minted@regoptg{#2}}
  \define@booleankey{minted@optlang}{#2}
    {\@namedef{minted@optlang\minted@lang @#2}{#3}\minted@regoptlang{#2}}
    {\@namedef{minted@optlang\minted@lang @#2}{#1}\minted@regoptlang{#2}}
  \define@booleankey{minted@optcmd}{#2}
    {\@namedef{minted@optcmd@#2}{#3}\minted@regoptcmd{#2}}
    {\@namedef{minted@optcmd@#2}{#1}\minted@regoptcmd{#2}}
}
```

\minted@define@extra Extra options are passed on to fancyvrb via Pygments.

```
\newcommand{\minted@define@extra}[1]{
  \define@key{minted@optg}{#1}{%
    \expandafter\def\expandafter\minted@optg@extra\expandafter{%
      \minted@optg@extra,#1=##1}}
  \@namedef{minted@optg@extra}{}
  \define@key{minted@optlang}{#1}{%
    \ifcsname minted@optlang\minted@lang @extra\endcsname\else
      \expandafter\gdef\csname minted@optlang\minted@lang @extra\endcsname{}%
    \fi
```

```
    \expandafter\let\expandafter\minted@optlang@extra%
      \csname minted@optlang\minted@lang @extra \endcsname
    \expandafter\def\expandafter\minted@optlang@extra\expandafter{%
      \minted@optlang@extra,#1=##1}%
    \expandafter\let\csname minted@optlang\minted@lang @extra\endcsname%
      \minted@optlang@extra
    \let\minted@optlang@extra\@empty}%
  \@namedef{minted@optlang@extra}{}
  \define@key{minted@optcmd}{#1}{%
    \expandafter\def\expandafter\minted@optcmd@extra\expandafter{%
      \minted@optcmd@extra,#1=##1}}
  \@namedef{minted@optcmd@extra}{}
}
```

\minted@define@extra@switch Extra switch options are also passed on to fancyvrb.

```
\newcommand{\minted@define@extra@switch}[1]{
  \define@booleankey{minted@optg}{#1}
    {\expandafter\def\expandafter\minted@optg@extra\expandafter{%
      \minted@optg@extra,#1}}
    {\expandafter\def\expandafter\minted@optg@extra\expandafter{%
      \minted@optg@extra,#1=false}}
  \define@booleankey{minted@optlang}{#1}
    {%
      \ifcsname minted@optlang\minted@lang @extra\endcsname\else
        \expandafter\gdef\csname minted@optlang\minted@lang @extra\endcsname{}%
      \fi
      \expandafter\let\expandafter\minted@optlang@extra%
        \csname minted@optlang\minted@lang @extra\endcsname
      \expandafter\def\expandafter\minted@optlang@extra\expandafter{%
        \minted@optlang@extra,#1}%
      \expandafter\let\csname minted@optlang\minted@lang @extra\endcsname%
        \minted@optlang@extra
      \let\minted@optlang@extra\@empty}
    {%
      \ifcsname minted@optlang\minted@lang @extra\endcsname\else
        \expandafter\gdef\csname minted@optlang\minted@lang @extra\endcsname{}%
      \fi
      \expandafter\let\expandafter\minted@optlang@extra%
        \csname minted@optlang\minted@lang @extra\endcsname
      \expandafter\def\expandafter\minted@optlang@extra\expandafter{%
        \minted@optlang@extra,#1=false}%
      \expandafter\let\csname minted@optlang\minted@lang @extra\endcsname%
        \minted@optlang@extra
      \let\minted@optlang@extra\@empty}
  \define@booleankey{minted@optcmd}{#1}
    {\expandafter\def\expandafter\minted@optcmd@extra\expandafter{%
      \minted@optcmd@extra,#1}}
    {\expandafter\def\expandafter\minted@optcmd@extra\expandafter{%
      \minted@optcmd@extra,#1=false}}
```

```
}
```

Actual option definitions.

Lexers.

```
% The following duplicates the 'extra' version; any difference?
%\minted@define@opt{tabsize}{-P tabsize=#1}
\minted@define@opt{encoding}{-P encoding=#1}
\minted@define@opt{outencoding}{-P outencoding=#1}
% Python console
\minted@define@switch{python3}{-P python3=True}
% PHP
\minted@define@switch[-P funcnamehighlighting=False]%
  {funcnamehighlighting}{-P funcnamehighlighting}
\minted@define@switch{startinline}{-P startinline}
```

Filters.

```
\minted@define@opt{gobble}{-F gobble:n=#1}
\minted@define@opt{codetagify}{-F codetagify:codetags=#1}
\minted@define@opt{keywordcase}{-F keywordcase:case=#1}
```

LaTeX formatter.

```
\minted@define@switch{texcl}{-P texcomments}
\minted@define@switch{texcomments}{-P texcomments}
\minted@define@switch{mathescape}{-P mathescape}
\minted@define@switch{linenos}{-P linenos}
\minted@define@optstyle
```

fancyvrb (via LaTeXformatter).

```
\minted@define@extra{frame}
\minted@define@extra{framesep}
\minted@define@extra{framerule}
\minted@define@extra{rulecolor}
\minted@define@extra{numbersep}
\minted@define@extra{numbers}
\minted@define@extra{firstnumber}
\minted@define@extra{stepnumber}
\minted@define@extra{firstline}
\minted@define@extra{lastline}
\minted@define@extra{baselinestretch}
\minted@define@extra{xleftmargin}
\minted@define@extra{xrightmargin}
\minted@define@extra{fillcolor}
\minted@define@extra{tabsize}
\minted@define@extra{fontfamily}
\minted@define@extra{fontsize}
\minted@define@extra{fontshape}
```

```
\minted@define@extra{fontseries}
\minted@define@extra{formatcom}
\minted@define@extra{label}
\minted@define@extra@switch{numberblanklines}
\minted@define@extra@switch{showspaces}
\minted@define@extra@switch{resetmargins}
\minted@define@extra@switch{samepage}
\minted@define@extra@switch{showtabs}
\minted@define@extra@switch{obeytabs}
```

Other options.

The old `bgcolor` is retained for compatibility, but in many cases a dedicated framing package may be preferable.

```
\let\minted@optcmd@bgcolor\@empty
\define@key{minted@optcmd}{bgcolor}{\@namedef{minted@optcmd@bgcolor}{#1}}
```

8.6 Internal helpers

`\minted@bgbox` Define an environment that may be wrapped around a `minted` environment to assign a background color. This is retained as a holdover from version 1.0. In most cases, it is probably better to use a dedicated framing package, such as mdframed or tcolorbox.

First, we need to define a new save box.

```
\newsavebox{\minted@bgbox}
```

Now we can define the environment that captures a code fragment inside a minipage and applies a background color.

```
\newenvironment{minted@colorbg}[1]{
    %\setlength{\fboxsep}{-\fboxrule}
    \def\minted@bgcol{#1}
    \noindent
    \begin{lrbox}{\minted@bgbox}
    \begin{minipage}{\linewidth-2\fboxsep}}
 {\end{minipage}
    \end{lrbox}%
    \colorbox{\minted@bgcol}{\usebox{\minted@bgbox}}}
```

`\minted@code` Create a file handle for saving code (and anything else that must be written to temp files).

```
\newwrite\minted@code
```

`\minted@savecode` Save code to be pygmentized to a file.

```
\newcommand{\minted@savecode}[1]{
  \immediate\openout\minted@code\jobname.pyg
  \immediate\write\minted@code{\expandafter\detokenize\expandafter{#1}}%
  \immediate\closeout\minted@code}
```

`\minted@FVB@VerbatimOut` We need a custom version of fancyvrb's `\FVB@VerbatimOut` that supports Unicode (everything written to file is `\detokenized`).

```
\newcommand{\minted@write@detok}[1]{%
  \immediate\write\FV@OutFile{\detokenize{#1}}}
\newcommand{\minted@FVB@VerbatimOut}[1]{%
  \@bsphack
  \begingroup
    \FV@UseKeyValues
    \FV@DefineWhiteSpace
    \def\FV@Space{\space}%
    \FV@DefineTabOut
    \let\FV@ProcessLine\minted@write@detok
    \immediate\openout\FV@OutFile #1\relax
    \let\FV@FontScanPrep\relax
    \let\@noligs\relax
    \FV@Scan}
```

`\minted@pygmentize` Pygmentize a file (default: `\jobname.pyg`) using the options provided.

Unfortunately, the logic for caching is a little complex due to operations that are OS- and engine-dependent.

The name of cached files is the result of concatenating the md5 of the code and the md5 of the command. This results in a filename that is longer than ideal (64 characters plus path and extension). Unfortunately, this is the only robust approach that is possible using the built-in pdfTeX hashing capabilities.[4] LuaTeX could do better, by hashing the command and code together. The Python script that provides XeTeX capabilities simply runs both the command and the code through a single sha1 hasher, but has the additional overhead of the `\write18` call and Python execution.

One potential concern is that caching should also keep track of the command from which code originates. What if identical code is highlighted with identical settings in both the `minted` environment and `\mintinline` command? In both cases, what is actually saved by Pygments is identical. The difference in final appearance is due to how the environment and command treat the Pygments output.

Notice that the `verboptions` macros don't need separating commas, since they're assembled in such a way that they will always have a leading comma.

[4] It would be possible to use only the cache of the code, but that approach breaks down as soon as the code is used multiple times with different options. While that may seem unlikely in practice, it occurs in this documentation and may be expected to occur in other docs.

```
\newcommand{\minted@pygmentize}[2][\jobname.pyg]{%
  \minted@checklang
  \def\minted@cmd{pygmentize -l #2 -f latex -F tokenmerge
    \minted@optg \space \csname minted@optlang\minted@lang\endcsname
    \space \minted@optcmd \space -P "verboptions=\minted@getoptg{extra}%
      \minted@getoptlang{extra}\minted@getoptcmd{extra}"
    -o \minted@infile \space #1}%
  % For debugging, uncomment:
  % \immediate\typeout{\minted@cmd}%
  \ifthenelse{\boolean{minted@cache}}%
    {%
      \expandafter\ifx\csname XeTeXinterchartoks\endcsname\relax
        \edef\minted@hash{\pdf@filemdfivesum{#1}\pdf@mdfivesum{\minted@cmd}}%
      \else
        \immediate\openout\minted@code\jobname.mintedcmd
        \immediate\write\minted@code{\minted@cmd}%
        \immediate\closeout\minted@code
        \immediate\write18{python minted-xetex-hasher.py "#1"}
        \input{\jobname.mintedmd5}
      \fi
      \ifwindows
        \edef\minted@infile{%
          \minted@cachedir\@backslashchar\minted@hash.pygtex}%
      \else
        \edef\minted@infile{%
          \minted@cachedir/\minted@hash.pygtex}%
      \fi
      \IfFileExists{\minted@cachedir/\minted@hash.pygtex}{}{%
        \immediate\write18{\minted@cmd}}%
      \expandafter\minted@addcachefile\expandafter{\minted@hash}%
      \minted@inputpyg}%
    {\immediate\write18{\minted@cmd}%
      \minted@inputpyg}%
}
\newcommand{\minted@inputpyg}{%
  \ifthenelse{\equal{\minted@optcmd@bgcolor}{}}%
    {}%
    {\begin{minted@colorbg}{\minted@optcmd@bgcolor}}%
  \input{\minted@infile}%
  \ifthenelse{\equal{\minted@optcmd@bgcolor}{}}%
    {}%
    {\end{minted@colorbg}}%
}
```

We need a way to have line counters on a per-language basis.

\minted@langlinenoson

```
\newcounter{minted@FancyVerbLineTemp}
```

```
\newcommand{\minted@langlinenoson}{%
  \ifcsname c@minted@lang\minted@lang\endcsname\else
    \newcounter{minted@lang\minted@lang}%
  \fi
  \setcounter{minted@FancyVerbLineTemp}{\value{FancyVerbLine}}%
  \setcounter{FancyVerbLine}{\value{minted@lang\minted@lang}}%
}
```

\minted@langlinenosoff

```
\newcommand{\minted@langlinenosoff}{%
  \setcounter{minted@lang\minted@lang}{\value{FancyVerbLine}}%
  \setcounter{FancyVerbLine}{\value{minted@FancyVerbLineTemp}}%
}
```

Disable the language-specific settings if the package option isn't used.

```
\ifthenelse{\boolean{minted@langlinenos}}{}{%
  \let\minted@langlinenoson\relax
  \let\minted@langlinenosoff\relax
}
```

8.7 Public API

\setminted Set global or language-level options.

```
\newcommand{\setminted}[2][]{%
  \ifthenelse{\equal{#1}{}}%
    {\setkeys{minted@optg}{#2}}%
    {\def\minted@lang{#1}\setkeys{minted@optlang}{#2}}%
}
```

\usemintedstyle Set style. This is a holdover from version 1, since `\setminted` can now accomplish this, and a hierarchy of style settings are now possible.

```
\newcommand{\usemintedstyle}[2][]{\setminted[#1]{style=#2}}
```

\mintinline Define an inline command. This requires some catcode acrobatics. The typical verbatim methods are not used. Rather, a different approach is taken that is generally more robust when used within other commands (for example, when used in footnotes).

Pygments saves code wrapped in a `Verbatim` environment. Getting the inline command to work correctly require redefining `Verbatim` to be `BVerbatim` temporarily. This approach would break if `BVerbatim` were ever redefined elsewhere.

```
\newcommand{\mintinline}[2][]{%
  \minted@resetoptcmd
```

```
  \setkeys{minted@optcmd}{#1}%
  \def\minted@lang{#2}%
  \begingroup
  \let\do\@makeother\dospecials
  \catcode`\{=1
  \catcode`\}=2
  \catcode`\^^I=\active
  \@ifnextchar\bgroup
    {\minted@inline@iii}%
    {\catcode`\{=12\catcode`\}=12
      \minted@inline@i}}
\def\minted@inline@i#1{%
  \endgroup
  \def\minted@inline@ii##1#1{%
    \minted@inline@iii{##1}}%
  \begingroup
  \let\do\@makeother\dospecials
  \minted@inline@ii}
\newcommand{\minted@inline@iii}[1]{%
  \endgroup
  \immediate\openout\minted@code\jobname.pyg
  \immediate\write\minted@code{\detokenize{#1}}%
  \immediate\closeout\minted@code
  \begingroup
  \RecustomVerbatimEnvironment{Verbatim}{BVerbatim}{}%
  \minted@pygmentize{\minted@lang}%
  \endgroup}
```

`\mint` Highlight a small piece of verbatim code.

```
\newcommand{\mint}[3][]{%
  \def\minted@lang{#2}%
  \DefineShortVerb{#3}%
  \minted@resetoptcmd
  \setkeys{minted@optcmd}{#1}%
  \SaveVerb[aftersave={%
    \UndefineShortVerb{#3}%
    \minted@savecode{\FV@SV@minted@verb}%
    \minted@langlinenoson
    \minted@pygmentize{#2}
    \minted@langlinenosoff}]{minted@verb}#3}
```

`minted` Highlight a longer piece of code inside a verbatim environment.

```
\newenvironment{minted}[2][]
  {\VerbatimEnvironment
    \let\FVB@VerbatimOut\minted@FVB@VerbatimOut
    \def\minted@lang{#2}%
    \minted@resetoptcmd
```

```
    \setkeys{minted@optcmd}{#1}%
    \begin{VerbatimOut}[codes={\catcode`\^^I=12}]{\jobname.pyg}}%
{\end{VerbatimOut}%
    \minted@langlinenoson
    \minted@pygmentize{\minted@lang}%
    \minted@langlinenosoff}
```

`\inputminted` Highlight an external source file.

```
\newcommand{\inputminted}[3][]{%
  \def\minted@lang{#2}%
  \minted@resetoptcmd
  \setkeys{minted@optcmd}{#1}%
  \minted@pygmentize[#3]{#2}}
```

8.8 Command shortcuts

We allow the user to define shortcuts for the highlighting commands.

`\newminted` Define a new language-specific alias for the `minted` environment.

```
\newcommand{\newminted}[3][]{
```

First, we look whether a custom environment name was given as the first optional argument. If that's not the case, construct it from the language name (append "`code`").

```
  \ifthenelse{\equal{#1}{}}
    {\def\minted@envname{#2code}}
    {\def\minted@envname{#1}}
```

Now, we define two environments. The first takes no further arguments. The second, starred version, takes an extra argument that specifies option overrides.

```
  \newenvironment{\minted@envname}
    {\VerbatimEnvironment
      \begin{minted}[#3]{#2}}
    {\end{minted}}
  \newenvironment{\minted@envname *}[1]
    {\VerbatimEnvironment\begin{minted}[#3,##1]{#2}}
    {\end{minted}}}
```

`\newmint` Define a new language-specific alias for the `\mint` short form.

```
\newcommand{\newmint}[3][]{
```

Same as with `\newminted`, look whether an explicit name is provided. If not, take the language name as command name.

```
  \ifthenelse{\equal{#1}{}}
    {\def\minted@shortname{#2}}
    {\def\minted@shortname{#1}}
```

And define the macro.

```
  \expandafter\newcommand\csname\minted@shortname\endcsname[2][]{
    \mint[#3,##1]{#2}##2}}
```

`\newmintedfile` Define a new language-specific alias for `\inputminted`.

```
\newcommand{\newmintedfile}[3][]{
```

Here, the default macro name (if none is provided) appends "`file`" to the language name.

```
  \ifthenelse{\equal{#1}{}}
    {\def\minted@shortname{#2file}}
    {\def\minted@shortname{#1}}
```

. . . and define the macro.

```
  \expandafter\newcommand\csname\minted@shortname\endcsname[2][]{
    \inputminted[#3,##1]{#2}{##2}}}
```

`\newmintinline` Define an alias for `\mintinline`.

As is usual with inline commands, a little catcode trickery must be employed.

```
\newcommand{\newmintinline}[3][]{%
  \ifthenelse{\equal{#1}{}}%
    {\def\minted@shortname{#2inline}}%
    {\def\minted@shortname{#1}}%
    \expandafter\newcommand\csname\minted@shortname\endcsname{%
      \begingroup
      \let\do\@makeother\dospecials
      \catcode`\{=1
      \catcode`\}=2
      \@ifnextchar[{\endgroup\minted@inliner[#3][#2]}%
        {\endgroup\minted@inliner[#3][#2][]}}%
    \def\minted@inliner[##1][##2][##3]{\mintinline[##1,##3]{##2}}%
}
```

8.9 Float support

`listing` Define a new floating environment to use for floated listings.

```
\@ifundefined{minted@float@within}
  {\newfloat{listing}{h}{lol}}
  {\newfloat{listing}{h}{lol}[\minted@float@within]}
```

`\listingcaption` The name that is displayed before each individual listings caption and its number. The macro `\listingscaption` can be redefined by the user.

```
\newcommand{\listingscaption}{Listing}
```

The following definition should not be changed by the user.

```
\floatname{listing}{\listingscaption}
```

`\listoflistingscaption` The caption that is displayed for the list of listings.

```
\newcommand{\listoflistingscaption}{List of listings}
```

`\listoflistings` Used to produce a list of listings (like `\listoffigures` etc.). This may well clash with other packages (for example, listings) but we choose to ignore this since these two packages shouldn't be used together in the first place.

```
\providecommand{\listoflistings}{\listof{listing}{\listoflistingscaption}}
```

8.10 Epilogue

Check whether LaTeX was invoked with `-shell-escape` option, make sure `pygmentize` exists, and set the default style.

```
\AtEndOfPackage{
  \ifnum\pdf@shellescape=1\relax\else
    \PackageError{minted}
     {You must invoke LaTeX with the
      -shell-escape flag}
     {Pass the -shell-escape flag to LaTeX. Refer to the minted.sty
      documentation for more information.}%
  \fi
  \TestAppExists{pygmentize}
  \ifAppExists\else
    \PackageError{minted}
     {You must have 'pygmentize' installed
      to use this package}
     {Refer to the installation instructions in the minted
      documentation for more information.}
  \fi
  \setminted{style=default}%
}
```

8.11 Final cleanup

Clean up temp files. What actually needs to be done depends on caching and engine.

```
\AtEndDocument{
  \expandafter\ifx\csname XeTeXinterchartoks\endcsname\relax
  \else
    \DeleteFile{minted-xetex-hasher.py}
    \DeleteFile{\jobname.mintedcmd}
    \DeleteFile{\jobname.mintedmd5}
  \fi
  \DeleteFile{\jobname.pyg}%
}
```