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

v2.2 from 2016/06/08

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.

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 appearance can be customized with a number of options and color 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:

```
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 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. Pygments may also be installed with `pip`:

```
$ pip install Pygments
```

If you already have Pygments installed, be aware that the latest version is recommended (at least 1.4 or later). Some features, such as `escapeinside`, will only work with 2.0+. minted may work with versions as early as 1.2, but there are no guarantees.

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`
- `kvoptions`
- `fancyvrb`
- `float`
- `ifthen`
- `calc`
- `ifplatform`
- `pdftexcmds`
- `etoolbox`
- `xstring`
- `xcolor`
- `lineno`
- `framed`
- `shellesc (for luatex 0.87+)`

2.3 Installing minted

You can probably install minted with your TEX distribution's package manager. Otherwise, or if you want the absolute latest version, you can install it manually by following the directions below.

You may download `minted.sty` from the project's homepage. 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 Transitioning to version 2

Transitioning from minted 1.7 to 2.0+ should require no changes in almost all cases. Version 2 provides the same interface and all of the same features.

In cases when custom code was used to hook into the minted internals, it may still be desirable to use the old minted 1.7. For those cases, the new package minted1 is provided. Simply load this before any other package attempts to load minted, and you will have the code from 1.7.

A brief summary of new features in version 2.0 is provided below. More detail is available in the Version History.

- New inline command `\mintinline`.
- Support for caching highlighted code with new package option `cache`. This drastically reduces package overhead. Caching is on by default. A cache directory called `_minted-`⟨*document name*⟩ will be created in the document root directory. This may be modified with the `cachedir` package option.
- Automatic line breaking for all commands and environments with new option `breaklines`. Many additional options for customizing line breaking.
- Support for Unicode under the pdfTeX engine.
- Set document-wide options using `\setminted{`⟨*opts*⟩`}`. Set language-specific options using `\setminted[`⟨*lang*⟩`]{`⟨*opts*⟩`}`. Similarly, set inline-specific options using `\setmintedinline`.
- Package option `langlinenos`: do line numbering by language.
- Many new options, including `encoding`, `autogobble`, and `escapeinside` (requires Pygments 2.0+).
- New package option `outputdir` provides compatibility with command-line options `-output-directory` and `-aux-directory`.
- New package option `draft` disables Python use to give maximum performance.
- `\mint` can now take code delimited by matched curly braces `{}`.

4 Basic usage

4.1 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 it only with documents from trusted sources.

Working with OS X

If you are using minted with some versions/configurations of OS X, and are using caching with a large number of code blocks (>256), you may receive an error like

```
OSError: [Errno 24] Too many open files:
```

This is due to the way files are handled by the operating system, combined with the way that caching works. To resolve this, you may use the OS X commands `launchctl limit maxfiles` or `ulimit -n` to increase the number of files that may be used.

4.2 A minimal complete example

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

```latex
\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`:

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

4.3 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}
```

```
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
```

This typesets a single line of code using a command rather than an environment, so it saves a little typing, but its output is equivalent to that of the `minted` environment.

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. The ⟨*code*⟩ may also be delimited with matched curly braces `{}`, so long as ⟨*code*⟩ itself does not contain unmatched curly braces. 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]

[1]For example, `\mintinline` works in footnotes! The main exception is when the code

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

4.4 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 Pygments website.

4.5 Supported languages

Pygments supports over 300 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
```

5 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):

contains the percent % or hash # characters, or unmatched curly braces.

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

```
\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

Customizing the `listing` environment

By default, the `listing` environment is created using the float package. In that case, the `\listingscaption` and `\listoflistingscaption` macros described below may be used to customize the caption and list of listings. If minted is loaded with the `newfloat` option, then the `listing` environment will be created with the more powerful newfloat package instead. newfloat is part of caption, which provides many options for customizing captions.

When newfloat is used to create the `listing` environment, customization should be achieved using newfloat's `\SetupFloatingEnvironment` command. For example, the string “Listing” in the caption could be changed to “Program code” using

```
\SetupFloatingEnvironment{listing}{name=Program code}
```

And “List of Listings” could be changed to “List of Program Code” with

```
\SetupFloatingEnvironment{listing}{listname=List of Program Code}
```

Refer to the newfloat and caption documentation for additional information.

`\listingscaption` (Only applies when package option `newfloat` is not used.) 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` (Only applies when package option `newfloat` is not used.) Likewise, the caption of the listings list, "List of Listings," can be changed by redefining `\listoflistingscaption`:

```
\renewcommand{\listoflistingscaption}{List of Program Code}
```

6 Options

6.1 Package options

`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 chapter:

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

`cache=⟨boolean⟩` `(default: true)` 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 several chunks of code to highlight. To avoid this, the package provides a `cache` option. This is on by default.

The `cache` option creates a directory `_minted-`⟨*jobname*⟩ in the document's root directory (this may be customized with the `cachedir` option).[3] 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.[4]

`cachedir=⟨directory⟩` `(def: _minted-⟨jobname⟩)` This allows the directory in which cached files are stored to be specified. Paths

[3] The directory is actually named using a "sanitized" copy of ⟨*jobname*⟩, in which spaces and asterisks have been replaced by underscores, and double quotation marks have been stripped. If the file name contains spaces, `\jobname` will contain a quote-wrapped name, except under older versions of MiKTeX which used the name with spaces replaced by asterisks. Using a "sanitized" ⟨*jobname*⟩ is simpler than accomodating the various escaping conventions.

[4] 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.

should use forward spaces, even under Windows.

Special characters must be escaped. For example, `cachedir=~/mintedcache` would not work because the tilde `~` would be converted into the LaTeX commands for a non-breaking space, rather than being treated literally. Instead, use `\string~/mintedcache`, `\detokenize{~/mintedcache}`, or an equivalent solution.

Paths may contain spaces, but only if the entire ⟨*directory*⟩ is wrapped in curly braces `{}`, and only if the spaces are quoted. For example,

```
cachedir = {\detokenize{~/"minted cache"/"with spaces"}}
```

Note that the cache directory is relative to the `outputdir`, if an `outputdir` is specified.

`finalizecache=`⟨*boolean*⟩ (default: `false`)

In some cases, it may be desirable to use minted in an environment in which `-shell-escape` is not allowed. A document might be submitted to a publisher or preprint server or used with an online service that does not support `-shell-escape`. This is possible as long as minted content does not need to be modified.

Compiling with the `finalizecache` option prepares the cache for use in an environment without `-shell-escape`.[5] Once this has been done, the `finalizecache` option may be swapped for the `frozencache` option, which will then use the frozen (static) cache in the future, without needing `-shell-escape`.

`frozencache=`⟨*boolean*⟩ (default: `false`)

Use a frozen (static) cache created with the `finalizecache` option. When `frozencache` is on, `-shell-escape` is not needed, and Python and Pygments are not required. In addition, any external files accessed through `\inputminted` are no longer necessary.

This option must be used with care. A document *must* be in final form, as far as minted is concerned, *before* `frozencache` is turned on, and the document *must* have been compiled with `finalizecache`. When this option is on, minted content cannot be modified, except by editing the cache files directly. Changing any minted settings that require Pygments or Python is not possible. If minted content is incorrectly modified after `frozencache` is turned on, minted *cannot* detect the modification.

If you are using `frozencache`, and want to verify that minted settings or content have not been modified in an invalid fashion, you can test the cache using the following procedure.

1. Obtain a copy of the cache used with `frozencache`.

[5] Ordinarily, cache files are named using an MD5 hash of highlighting settings and highlighted text. `finalizecache` renames cache files using a `listing<number>.pygtex` scheme. This makes it simpler to match up document content and cache files, and is also necessary for the `XeTeX` engine since it lacks the built-in MD5 capabilities that `pdfTeX` and `LuaTeX` have.

2. Compile the document in an environment that supports `-shell-escape`, with `finalizecache=true` and `frozencache=false`. This essentially regenerates the frozen (static) cache.

3. Compare the original cache with the newly generated cache. Under Linux and OS X, you could use `diff`; under Windows, you probably want `fc`. If minted content and settings have not been modified in an invalid fashion, all files will be identical (assuming that compatible versions of Pygments are used for both caches).

`draft=`⟨*boolean*⟩ `(default: false)`

This uses fancyvrb alone for all typesetting; Pygments is not used. This trades syntax highlighting and some other minted features for faster compiling. Performance should be essentially the same as using fancyvrb directly; no external temporary files are used. Note that if you are not changing much code between compiles, the difference in performance between caching and draft mode may be minimal. Also note that `draft` settings are typically inherited from the document class.

Draft mode does not support `autogobble`. Regular `gobble`, `linenos`, and most other options not related to syntax highlighting will still function in draft mode.

Documents can usually be compiled without shell escape in draft mode. The ifplatform package may issue a warning about limited functionality due to shell escape being disabled, but this may be ignored in almost all cases. (Shell escape is only really required if you have an unusual system configuration such that the `\ifwindows` macro must fall back to using shell escape to determine the system. See the ifplatform documentation for more details: http://www.ctan.org/pkg/ifplatform.)

If the `cache` option is set, then all existing cache files will be kept while draft mode is on. This allows caching to be used intermitently with draft mode without requiring that the cache be completely recreated each time. Automatic cleanup of cached files will resume as soon as draft mode is turned off. (This assumes that the auxiliary file has not been deleted in the meantime; it contains the cache history and allows automatic cleanup of unused files.)

`final=`⟨*boolean*⟩ `(default: true)`

This is the opposite of `draft`; it is equivalent to `draft=false`. Again, note that `draft` and `final` settings are typically inherited from the document class.

`kpsewhich=`⟨*boolean*⟩ `(default: false)`

This option uses `kpsewhich` to locate files that are to be highlighted. Some build tools such as `texi2pdf` function by modifying `TEXINPUTS`; in some cases, users may customize `TEXINPUTS` as well. The `kpsewhich` option allows minted to work with such configurations.

This option may add a noticeable amount of overhead on some systems, or with some system configurations.

This option does *not* make minted work with the `-output-directory` and `-aux-directory` command-line options for LaTeX. For those, see the `outputdir` package option.

Under Windows, this option currently requires that PowerShell be installed. It may need to be installed in versions of Windows prior to Windows 7.

`langlinenos=`⟨*boolean*⟩ `(default: false)`

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}
```

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

1  def func
2      puts "message"
3  end

3  def g(x):
4      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. Rather, it would pick up with the Ruby line numbering.

`newfloat=`⟨*boolean*⟩ `(default: false)`

By default, the `listing` environment is created using the float package. The `newfloat` option creates the environment using newfloat instead. This provides better integration with the caption package.

`outputdir=`⟨*directory*⟩ `(default:` ⟨*none*⟩`)`

The `-output-directory` and `-aux-directory` (MiKTeX) command-line options for LaTeX cause problems for minted, because the minted temporary files are saved in `<outputdir>`, but minted still looks for them in the document root directory. There is no way to access the value of the command-line option so that minted can automatically look in the right place. But it is possible to allow the output directory to be specified manually as a package option.

The output directory should be specified using an absolute path or a path relative to the document root directory. Paths should use forward spaces, even under

Windows. Special characters must be escaped, while spaces require quoting and need the entire ⟨*directory*⟩ to be wrapped in curly braces `{}`. See `cachedir` above for examples of escaping and quoting.

`section` To control how LaTeX counts the `listing` floats, you can pass either the `section` or `chapter` option when loading the minted package.

6.2 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. There is also an `autogobble` option that detects the length of this whitespace automatically.

```
\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[`⟨*language*⟩`]{`⟨*key=value,...*⟩`}`. Language-specific options override document-wide options. Individual command and environment options override language-specific options.

`\setmintedinline` You may wish to set separate options for `\mintinline`, either for the document as a whole or for a specific language. This is possible via `\setmintedinline`. The syntax is `\setmintedinline[`⟨*language*⟩`]{`⟨*key=value,...*⟩`}`. Language-specific options override document-wide options. Individual command options override language-specific options. All settings specified with `\setmintedinline` override those set with `\setminted`. That is, inline settings always have a higher precedence than general settings.

6.3 Available options

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

`autogobble` (boolean) (default: `false`)
Remove (gobble) all common leading whitespace from code. Essentially a version of `gobble` that automatically determines what should be removed. Good for code that originally is not indented, but is manually indented after being pasted into a LaTeX document.

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

...text.

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

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

`breakafter` (string) (default: ⟨*none*⟩)
Break lines after specified characters, not just at spaces, when `breaklines=true`. For example, `breakafter=-/` would allow breaks after any hyphens or slashes. Special characters given to `breakafter` should be backslash-escaped (usually `#`, `{`, `}`, `%`, `[`, `]`; the backslash \ may be obtained via `\\`).

For an alternative, see breakbefore. When breakbefore and breakafter are used for the same character, breakbeforegroup and breakaftergroup must both have the same setting.

```
\begin{minted}[breaklines, breakafter=d]{python}
some_string = 'SomeTextThatGoesOnAndOnForSoLongThatItCouldNeverFitOnOneLine'
\end{minted}
```

```
some_string = 'SomeTextThatGoesOnAndOnForSoLongThatItCould⌋
 ↪  NeverFitOnOneLine'
```

breakaftergroup (boolean) (default: true)
When breakafter is used, group all adjacent identical characters together, and only allow a break after the last character. When breakbefore and breakafter are used for the same character, breakbeforegroup and breakaftergroup must both have the same setting.

breakaftersymbolpre (string) (default: \,\footnotesize\ensuremath{_\rfloor}, ⌋)
The symbol inserted pre-break for breaks inserted by breakafter.

breakaftersymbolpost (string) (default: ⟨*none*⟩)
The symbol inserted post-break for breaks inserted by breakafter.

breakanywhere (boolean) (default: false)
Break lines anywhere, not just at spaces, when breaklines=true.

```
\begin{minted}[breaklines, breakanywhere]{python}
some_string = 'SomeTextThatGoesOnAndOnForSoLongThatItCouldNeverFitOnOneLine'
\end{minted}
```

```
some_string = 'SomeTextThatGoesOnAndOnForSoLongThatItCouldNev⌋
 ↪  erFitOnOneLine'
```

breakanywheresymbolpre (string) (default: \,\footnotesize\ensuremath{_\rfloor}, ⌋)
The symbol inserted pre-break for breaks inserted by breakanywhere.

breakanywheresymbolpost (string) (default: ⟨*none*⟩)
The symbol inserted post-break for breaks inserted by breakanywhere.

breakautoindent (boolean) (default: true)
When a line is broken, automatically indent the continuation lines to the indentation level of the first line. When breakautoindent and breakindent are used together, the indentations add. This indentation is combined with breaksymbolindentleft to give the total actual left indentation. Does not apply to \mintinline.

breakbefore (string) (default: ⟨*none*⟩)
Break lines before specified characters, not just at spaces, when breaklines=true. For example, breakbefore=A would allow breaks before capital A's. Special

characters given to `breakbefore` should be backslash-escaped (usually `#`, `{`, `}`, `%`, `[`, `]`; the backslash `\` may be obtained via `\\`).

For an alternative, see `breakafter`. When `breakbefore` and `breakafter` are used for the same character, `breakbeforegroup` and `breakaftergroup` must both have the same setting.

```
\begin{minted}[breaklines, breakbefore=A]{python}
some_string = 'SomeTextThatGoesOnAndOnForSoLongThatItCouldNeverFitOnOneLine'
\end{minted}
```

```
some_string = 'SomeTextThatGoesOn⌋
 ↪  AndOnForSoLongThatItCouldNeverFitOnOneLine'
```

`breakbeforegroup` (boolean) (default: `true`)
When `breakbefore` is used, group all adjacent identical characters together, and only allow a break before the first character. When `breakbefore` and `breakafter` are used for the same character, `breakbeforegroup` and `breakaftergroup` must both have the same setting.

`breakbeforesymbolpre` (string) (default: `\,\footnotesize\ensuremath{_\rfloor}`, ⌋)
The symbol inserted pre-break for breaks inserted by `breakbefore`.

`breakbeforesymbolpost` (string) (default: ⟨*none*⟩)
The symbol inserted post-break for breaks inserted by `breakbefore`.

`breakbytoken` (boolean) (default: `false`)
Only break lines at locations that are not within tokens; prevent tokens from being split by line breaks. By default, `breaklines` causes line breaking at the space nearest the margin. While this minimizes the number of line breaks that are necessary, it can be inconvenient if a break occurs in the middle of a string or similar token.

This is not compatible with `draft` mode. A complete list of Pygments tokens is available at `http://pygments.org/docs/tokens/`. If the breaks provided by `breakbytoken` occur in unexpected locations, it may indicate a bug or shortcoming in the Pygments lexer for the language.

`breakbytokenanywhere` (boolean) (default: `false`)
Like `breakbytoken`, but also allows line breaks between immediately adjacent tokens, not just between tokens that are separated by spaces. Using `breakbytokenanywhere` with `breakanywhere` is redundant.

`breakindent` (dimension) (default: `0pt`)
When a line is broken, indent the continuation lines by this amount. When `breakautoindent` and `breakindent` are used together, the indentations add. This indentation is combined with `breaksymbolindentleft` to give the total actual left indentation. Does not apply to `\mintinline`.

breaklines (boolean) (default: false)

Automatically break long lines in minted environments and \mint commands, and wrap longer lines in \mintinline.

This is not compatible with the option obeytabs. Additional information about the incompatibility is on GitHub.

By default, automatic breaks occur at space characters. Use breakanywhere to enable breaking anywhere; use breakbytoken, breakbytokenanywhere, and breakafter for more fine-tuned breaking. Using escapeinside to escape to LaTeX and then insert a manual break is also an option. For example, use escapeinside=||, and then insert |\\| at the appropriate point. (Note that escapeinside does not work within strings.)

```
...text.
\begin{minted}[breaklines]{python}
def f(x):
    return 'Some text ' + str(x)
\end{minted}
```

...text.

```
def f(x):
    return 'Some text ' +
     ↪  str(x)
```

Breaking in minted and \mint may be customized in several ways. To customize the indentation of broken lines, see breakindent and breakautoindent. To customize the line continuation symbols, use breaksymbolleft and breaksymbolright. To customize the separation between the continuation symbols and the code, use breaksymbolsepleft and breaksymbolsepright. To customize the extra indentation that is supplied to make room for the break symbols, use breaksymbolindentleft and breaksymbolindentright. Since only the left-hand symbol is used by default, it may also be modified using the alias options breaksymbol, breaksymbolsep, and breaksymbolindent. Note than none of these options applies to \mintinline, since they are not relevant in the inline context.

An example using these options to customize the minted environment is shown below. This uses the \carriagereturn symbol from the dingbat package.

```
\begin{minted}[breaklines,
               breakautoindent=false,
               breaksymbolleft=\raisebox{0.8ex}{
                 \small\reflectbox{\carriagereturn}},
               breaksymbolindentleft=0pt,
               breaksymbolsepleft=0pt,
               breaksymbolright=\small\carriagereturn,
               breaksymbolindentright=0pt,
               breaksymbolsepright=0pt]{python}
def f(x):
    return 'Some text ' + str(x) + ' some more text ' +
    ↪  str(x) + ' even more text that goes on for a
    ↪  while'
\end{minted}
```

```
def f(x):
    return 'Some text ' + str(x) + ' some more text ' +
str(x) + ' even more text that goes on for a while'
```

Automatic line breaks are limited with Pygments styles that use a colored background behind large chunks of text. This coloring is accomplished with `\colorbox`, which cannot break across lines. It may be possible to create an alternative to `\colorbox` that supports line breaks, perhaps with TikZ, but the author is unaware of a satisfactory solution. The only current alternative is to redefine `\colorbox` so that it does nothing. For example,

```
\AtBeginEnvironment{minted}{\renewcommand{\colorbox}[3][]{#3}}
```

uses the etoolbox package to redefine `\colorbox` within all `minted` environments.

Automatic line breaks will not work with `showspaces=true` unless you use `breakanywhere`. You may be able to change the definition of `\FV@Space` if you need this; see the fancyvrb implementation for details.

`breaksymbol` (string) (default: `breaksymbolleft`)
Alias for `breaksymbolleft`.

`breaksymbolleft` (string) (default: `\tiny\ensuremath{\hookrightarrow}`, ↪)
The symbol used at the beginning (left) of continuation lines when `breaklines=true`. To have no symbol, simply set `breaksymbolleft` to an empty string ("`=,`" or "`={}`"). The symbol is wrapped within curly braces `{}` when used, so there is no danger of formatting commands such as `\tiny` "escaping."

The `\hookrightarrow` and `\hookleftarrow` may be further customized by the use of the `\rotatebox` command provided by graphicx. Additional arrow-type symbols that may be useful are available in the dingbat (`\carriagereturn`) and mnsymbol (hook and curve arrows) packages, among others.

Does not apply to `\mintinline`.

`breaksymbolright` (string) (default: ⟨*none*⟩)

The symbol used at breaks (right) when `breaklines=true`. Does not appear at the end of the very last segment of a broken line.

`breaksymbolindent` (dimension) (default: `breaksymbolindentleft`)
Alias for `breaksymbolindentleft`.

`breaksymbolindentleft` (dimension) (default: width of 4 characters in teletype font at default point size)
The extra left indentation that is provided to make room for `breaksymbolleft`. This indentation is only applied when there is a `breaksymbolleft`.

This may be set to the width of a specific number of (fixed-width) characters by using an approach such as

```
\newdimen\temporarydimen
\settowidth{\temporarydimen}{\ttfamily aaaa}
```

and then using `breaksymbolindentleft=\temporarydimen`.

Does not apply to `\mintinline`.

`breaksymbolindentright` (dimension) (default: width of 4 characters in teletype font at default point size)
The extra right indentation that is provided to make room for `breaksymbolright`. This indentation is only applied when there is a `breaksymbolright`.

`breaksymbolsep` (dimension) (default: `breaksymbolsepleft`)
Alias for `breaksymbolsepleft`

`breaksymbolsepleft` (dimension) (default: `1em`)
The separation between the `breaksymbolleft` and the adjacent code. Does not apply to `\mintinline`.

`breaksymbolsepright` (dimension) (default: `1em`)
The separation between the `breaksymbolright` and the adjacent code.

`bgcolor` (string) (default: ⟨*none*⟩)
Background color of the listing. Be aware that this option has several limitations (described below); see “Framing alternatives” below for more powerful alternatives.

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";
?>
```

This option puts `minted` environments and `\mint` commands in a `snugshade*` environment from the framed package, which supports breaks across pages. (Prior to minted 2.2, a `minipage` was used, which prevented page breaks and gave undesirable spacing from surrounding text.) Be aware that if `bgcolor` is used with `breaklines=true`, and a line break occurs just before a page break, then text may extend below the colored background in some instances. It is best to use a more advanced framing package in those cases; see "Framing alternatives" below.

This option puts `\mintinline` inside a `\colorbox`, which **does not allow line breaks**. If you want to use `\setminted` to set background colors, and only want background colors on `minted` and `\mint`, you may use `\setmintedinline{bgcolor={}}` to turn off the coloring for inline commands.

Framing alternatives

If you want more reliable and advanced options for background colors and framing, you should consider a more advanced framing package such as mdframed or tcolorbox. It is easy to add framing to minted commands and environments using the etoolbox package, which is automatically loaded by minted. For example, using mdframed:

```
\BeforeBeginEnvironment{minted}{\begin{mdframed}}
\AfterEndEnvironment{minted}{\end{mdframed}}
```

Some framing packages also provide built-in commands for such purposes. For example, mdframed provides a `\surroundwithmdframed` command, which could be used to add a frame to all `minted` environments:

```
\surroundwithmdframed{minted}
```

tcolorbox even provides a built-in framing environment with minted support. Simply use `\tcbuselibrary{minted}` in the preamble, and then put code within a `tcblisting` environment:

```
\begin{tcblisting}{<tcb options>,
                   minted language=<language>,
                   minted style=<style>,
                   minted options={<option list>} }
<code>
\end{tcblisting}
```

tcolorbox provides other commands and environments for fine-tuning listing appearance and for working with external code files.

`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. See also `outencoding`.

`escapeinside` (string) (default: ⟨*none*⟩)
Escape to LaTeX between the two characters specified in `(string)`. All code between the two characters will be interpreted as LaTeX and typeset accordingly. This allows for additional formatting. The escape characters need not be identical.

Special LaTeX characters must be escaped when they are used as the escape characters (for example, `escapeinside=\#\%`). Requires Pygments 2.0+.

Escaping does not work inside strings and comments (for comments, there is `texcomments`). As of Pygments 2.0.2, this means that escaping is "fragile" with some lexers. Due to the way that Pygments implements `escapeinside`, any "escaped" LaTeX code that resembles a string or comment for the current lexer may break `escapeinside`. There is a Pygments issue for this case. Additional details and a limited workaround for some scenarios are available on the minted GitHub site.

```
\begin{minted}[escapeinside=||]{py}
def f(x):
    y = x|\colorbox{green}{**}|2
    return y
\end{minted}
```

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

Note that when math is used inside escapes, in a few cases ligature handling may need to be modified. The single-quote character (`'`) is normally a shortcut for `^\prime` in math mode, but this is disabled in verbatim content as a byproduct of ligatures being disabled. For the same reason, any package that relies on active characters in math mode (for example, icomma) will produce errors along the lines of `TeX capacity exceeded` and `\leavevmode\kern\z@`. This may be fixed by modifying `\@noligs`, as described at `http://tex.stackexchange.com/questions/223876`. minted currently does not attempt to patch `\@noligs` due to the potential for package conflicts.

`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) (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 5) 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. Usage as in package listings. See the note under `escapeinside` regarding math and ligatures.

`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`)
Due to the many issues with fancyvrb's implementation of `obeytabs`, this option should be avoided if possible.

Treat tabs as tabs instead of converting them to spaces.

This is not compatible with the option `breaklines`.

This will cause errors with tabbed indentation inside multiline comments.

There is a GitHub issue with additional technical details.

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

`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.

`stripall` (boolean) (default: `false`)
Strip all leading and trailing whitespace from the input.

`stripnl` (boolean) (default: `true`)

Strip leading and trailing newlines from the input.

`tabsize` (integer) (default: `8`)
The number of spaces a tab is equivalent to. If `obeytabs` is *not* active, tabs will be converted into this number of spaces. If `obeytabs` is active, tab stops will be set this number of space characters apart.

`texcl` (boolean) (default: `false`)
Enables LaTeX code inside comments. Usage as in package listings. See the note under `escapeinside` regarding math and ligatures.

`texcomments` (boolean) (default: `false`)
Enables LaTeX code inside comments. The newer name for `texcl`. See the note under `escapeinside` regarding math and ligatures.

As of Pygments 2.0.2, `texcomments` fails with multiline C/C++ preprocessor directives, and may fail in some other circumstances. This is because preprocessor directives are tokenized as `Comment.Preproc`, so `texcomments` causes preprocessor directives to be treated as literal LaTeX code. An issue has been opened at the Pygments site; additional details are also available on the minted GitHub site.

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

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

7 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:

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

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

```cpp
template <typename T>
T id(T value) {
    return value;
}
```

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*⟩`}`.

Like normal minted environments, environments created with `\newminted` may be used within other environment definitions. Since the minted environments use fancyvrb internally, any environment based on them must include the fancyvrb command `\VerbatimEnvironment`. This allows fancyvrb to determine the name of the environment that is being defined, and correctly find its end. It is best to include this command at the beginning of the definition. For example,

```
\newminted{cpp}{gobble=2,linenos}
\newenvironment{env}{\VerbatimEnvironment\begin{cppcode}}{\end{cppcode}}
```

`\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
```

X`my␣$foo␣=␣$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`.

8 FAQ and 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.

- **I receive a "Font Warning: Some font shapes were not available" message, or bold or italic seem to be missing.** This due to a limitation in the font that is currently in use for typesetting code. In some cases, the default font shapes that LaTeX substitutes are perfectly adequate, and the warning may be ignored. In other cases, the font substitutions may not clearly indicate bold or italic text, and you will want to switch to a different font. See The LaTeX Font Catalogue's section on Typewriter Fonts for alternatives. If you like the default LaTeX fonts, the lmodern package is a good place to start. The beramono and courier packages may also be good options.
- **I receive a "Too many open files" error under OS X when using caching.** See the note on OS X under Section 4.1.
- **Weird things happen when I use the fancybox package.** fancybox conflicts with fancyvrb, which minted uses internally. When using fancybox, make sure that it is loaded before minted (or before fancyvrb, if fancyvrb is not loaded by minted).
- **When I use minted with KOMA-Script document classes, I get warnings about `\float@addtolists`.** minted uses the float package to produce floated listings, but this conflicts with the way KOMA-Script does floats. Load the package scrhack to resolve the conflict. Or use minted's `newfloat` package option.
- **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}`, perhaps with `\usepackage{lmodern}`, or something similar.
- **I'm getting errors with math, something like `TeX capacity exceeded` and `\leavevmode\kern\z@`.** This is due to ligatures being disabled within verbatim content. See the note under `escapeinside`.
- **Quotation marks and backticks don't look right. Backtick characters ` are appearing as left quotes. Single quotes are appearing as curly right quotes.** This is due to how Pygments outputs LaTeX code, combined with how LaTeX deals with verbatim content. Try `\usepackage{upquote}`.

- **I'm getting errors with Beamer.** Due to how Beamer treats verbatim content, you may need to use either the `fragile` or `fragile=singleslide` options for frames that contain minted commands and environments. `fragile=singleslide` works best, but it disables overlays. `fragile` works by saving the contents of each frame to a temp file and then reusing them. This approach allows overlays, but will break if you have the string `\end{frame}` at the beginning of a line (for example, in a `minted` environment). To work around that, you can indent the content of the environment (so that the `\end{frame}` is preceded by one or more spaces) and then use the `gobble` or `autogobble` options to remove the indentation.

- **Tabs are eaten by Beamer.** This is due to a bug in Beamer's treatment of verbatim content. Upgrade Beamer or use the linked patch. Otherwise, try `fragile=singleslide` if you don't need overlays, or consider using `\inputminted` or converting the tabs into spaces.

- **I'm trying to create several new minted commands/environments, and want them all to have the same settings. I'm saving the settings in a macro and then using the macro when defining the commands/environments. But it's failing.** This is due to the way that keyval works (minted uses it to manage options). Arguments are not expanded. See this and this for more information. It is still possible to do what you want; you just need to expand the options macro before passing it to the commands that create the new commands/environments. An example is shown below. The `\expandafter` is the vital part.

```
\def\args{linenos,frame=single,fontsize=\footnotesize,style=bw}

\newcommand{\makenewmintedfiles}[1]{%
  \newmintedfile[inputlatex]{latex}{#1}%
  \newmintedfile[inputc]{c}{#1}%
}

\expandafter\makenewmintedfiles\expandafter{\args}
```

- **I want to use `\mintinline` in a context that normally doesn't allow verbatim content.** The `\mintinline` command will already work in many places that do not allow normal verbatim commands like `\verb`, so make sure to try it first. If it doesn't work, one of the simplest alternatives is to save your code in a box, and then use it later. For example,

```
\newsavebox\mybox
\begin{lrbox}{\mybox}
\mintinline{cpp}{std::cout}
\end{lrbox}

\commandthatdoesnotlikeverbatim{Text \usebox{\mybox}}
```

- **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 polyglossia package is doing undesirable things to code. (For example, adding extra space around colons in French.)** You may need to put your code within `\begin{english}...\end{english}`. This may done for all `minted` environments using etoolbox in the preamble:

```
\usepackage{etoolbox}
\BeforeBeginEnvironment{minted}{\begin{english}}
\AfterEndEnvironment{minted}{\end{english}}
```

- **Tabs are being turned into the character sequence `^^I`.** This happens when you use XeLaTeX. You need to use the `-8bit` command-line option so that tabs may be written correctly to temporary files. See `http://tex.stackexchange.com/questions/58732/how-to-output-a-tabulation-into-a-file` for more on XeLaTeX's handling of tab characters.

- **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`. Or better yet, use minted's `newfloat` package option, which provides better caption compatibility.

- **I need a listing environment that supports page breaks.** The built-in listing environment is a standard float; it doesn't support page breaks. You will probably want to define a new environment for long floats. For example,

```
\usepackage{caption}
\newenvironment{longlisting}{\captionsetup{type=listing}}{}
```

  With the caption package, it is best to use minted's `newfloat` package option. See `http://tex.stackexchange.com/a/53540/10742` for more on `listing` environments with page breaks.

- **I want to use a custom script/executable to access Pygments, rather than `pygmentize`.** Redefine `\MintedPygmentize`:

```
\renewcommand{\MintedPygmentize}{...}
```

- **I want to use the command-line option `-output-directory`, or MiKTeX's `-aux-directory`, but am getting errors.** Use the package option `outputdir` to specify the location of the output directory. Unfortunately, there is no way for minted to detect the output directory automatically.

- **I want extended characters in frame labels, but am getting errors.** This can happen with minted <2.0 and Python 2.7, due to a terminal encoding issue with Pygments. It should work with any version of Python with minted 2.0+, which processes labels internally and does not send them to Python.
- **`minted` environments have extra vertical space inside `tabular`.** It is possible to create a custom environment that eliminates the extra space. However, a general solution that behaves as expected in the presence of adjacent text remains to be found.
- **I'm receiving a warning from `lineno.sty` that "Command `\@parboxrestore` has changed."** This can happen when minted is loaded after csquotes. Try loading minted first. If you receive this message when you are not using csquotes, you may want to experiment with the order of loading packages and might also open an issue.

Acknowledgements

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

Geoffrey Poore: Thanks to Marco Daniel for the code on `tex.stackexchange.com` that inspired automatic line breaking. Thanks to Patrick Vogt for improving TikZ externalization compatibility.

Version History

v2.2 (2016/06/08)

- All uses of `\ShellEscape` (`\write18`) no longer wrap file names and paths with double quotes. This allows a cache directory to be specified relative to a user's home directory, for example, `~/minted_cache`. `cachedir` and `outputdir` paths containing spaces will now require explicit quoting of the parts of the paths that contain spaces, since `minted` no longer supplies quoting. See the updated documentation for examples (#89).
- Added `breakbefore`, `breakbeforegroup`, `breakbeforesymbolpre`, and `breakbeforesymbolpost`. These parallel `breakafter*`. It is possible to use `breakbefore` and `breakafter` for the same character, so long as `breakbeforegroup` and `breakaftergroup` have the same setting (#117).
- Added package options `finalizecache` and `frozencache`. These allow the cache to be prepared for (`finalizecache`) and then used

(frozencache) in an environment in which -shell-escape, Python, and/or Pygments are not available. Note that this only works if minted content does not need to be modified, and if no settings that depend on Pygments or Python need to be changed (#113).

- Style names containing hyphens and underscores (paraiso-light, paraiso-dark, algol_nu) now work (#111).
- The shellesc package is now loaded, when available, for compatibility with luatex 0.87+ (TeX Live 2016+, etc.). \ShellEscape is now used everywhere instead of \immediate\write18. If shellesc is not available, then a \ShellEscape macro is created. When shellesc is loaded, there is a check for versions before v0.01c to patch a bug in v0.01b (present in TeX Live 2015) (#112).
- The bgcolor option now uses the snugshade* environment from the framed package, so bgcolor is now compatible with page breaks. When bgcolor is in use, immediately preceding text will no longer push the minted environment into the margin, and there is now adequate spacing from surrounding text (#121).
- Added missing support for fancyvrb's labelposition (#102).
- Improved fix for TikZ externalization, thanks to Patrick Vogt (#73).
- Fixed breakautoindent; it was disabled in version 2.1 due to a bug in breakanywhere.
- Properly fixed handling of \MintedPygmentize (#62).
- Added note on incompatibility of breaklines and obeytabs options. Trying to use these together will now result in a package error (#99). Added note on issues with obeytabs and multiline comments (#88). Due to the various obeytabs issues, the docs now discourage using obeytabs.
- Added note to FAQ on fancybox and fancyvrb conflict (#87).
- Added note to docs on the need for \VerbatimEnvironment in environment definitions based on minted environments.

v2.1 (2015/09/09)

- Changing the highlighting style now no longer involves re-highlighing code. Style may be changed with almost no overhead.
- Improved control of automatic line breaks. New option breakanywhere allows line breaks anywhere when breaklines=true. The pre-break and post-break symbols for these types of breaks may be set with breakanywheresymbolpre and breakanywheresymbolpost (#79). New option breakafter allows specifying characters after which line breaks are allowed. Breaks between adjacent, identical characters may be controlled with breakaftergroup. The pre-break and post-break symbols for these types of breaks may be set with breakaftersymbolpre and breakaftersymbolpost.

- `breakbytoken` now only breaks lines between tokens that are separated by spaces, matching the documentation. The new option `breakbytokenanywhere` allows for breaking between tokens that are immediately adjacent. Fixed a bug in `\mintinline` that produced a following linebreak when `\mintinline` was the first thing in a paragraph and `breakbytoken` was true (#77).
- Fixed a bug in draft mode option handling for `\inputminted` (#75).
- Fixed a bug with `\MintedPygmentize` when a custom `pygmentize` was specified and there was no `pygmentize` on the default path (#62).
- Added note to docs on caching large numbers of code blocks under OS X (#78).
- Added discussion of current limitations of `texcomments` (#66) and `escapeinside` (#70).
- PGF/Ti*k*Z externalization is automatically detected and supported (#73).
- The package is now compatible with LaTeX files whose names contain spaces (#85).

v2.0 (2015/01/31)

- Added the compatibility package `minted1`, which provides the minted 1.7 code. This may be loaded when 1.7 compatibility is required. This package works with other packages that `\RequirePackage{minted}`, so long as it is loaded first.
- Moved all old `\changes` into `changelog`.

Development releases for 2.0 (2014–January 2015)

- Caching is now on by default.
- Fixed a bug that prevented compiling under Windows when file names contained commas.
- Added `breaksymbolleft`, `breaksymbolsepleft`, `breaksymbolindentleft`, `breaksymbolright`, `breaksymbolsepright`, and `breaksymbolindentright` options. `breaksymbol`, `breaksymbolsep`, and `breaksymbolindent` are now aliases for the correspondent `*left` options.
- Added `kpsewhich` package option. This uses `kpsewhich` to locate the files that are to be highlighted. This provides compatibility with build tools like `texi2pdf` that function by modifying `TEXINPUTS` (#25).
- Fixed a bug that prevented `\inputminted` from working with `outputdir`.
- Added informative error messages when Pygments output is missing.
- Added `final` package option (opposite of `draft`).

- Renamed the default cache directory to `_minted-<jobname>` (replaced leading period with underscore). The leading period caused the cache directory to be hidden on many systems, which was a potential source of confusion.
- `breaklines` and `breakbytoken` now work with `\mintinline` (#31).
- `bgcolor` may now be set through `\setminted` and `\setmintedinline`.
- When math is enabled via `texcomments`, `mathescape`, or `escapeinside`, space characters now behave as in normal math by vanishing, instead of appearing as literal spaces. Math need no longer be specially formatted to avoid undesired spaces.
- In default value of `\listoflistingscaption`, capitalized "Listings" so that capitalization is consistent with default values for other lists (figures, tables, algorithms, etc.).
- Added `newfloat` package option that creates the `listing` environment using `newfloat` rather than `float`, thus providing better compatibility with the `caption` package (#12).
- Added support for Pygments option `stripall`.
- Added `breakbytoken` option that prevents `breaklines` from breaking lines within Pygments tokens.
- `\mintinline` uses a `\colorbox` when `bgcolor` is set, to give more reasonable behavior (#57).
- For PHP, `\mintinline` automatically begins with `startinline=true` (#23).
- Fixed a bug that threw off line numbering in `minted` when `langlinenos=false` and `firstnumber=last`. Fixed a bug in `\mintinline` that threw off subsequent line numbering when `langlinenos=false` and `firstnumber=last`.
- Improved behavior of `\mint` and `\mintinline` in `draft` mode.
- The `\mint` command now has the additional capability to take code delimited by paired curly braces `{}`.
- It is now possible to set options only for `\mintinline` using the new `\setmintedinline` command. Inline options override options specified via `\setminted`.
- Completely rewrote option handling. fancyvrb options are now handled on the LaTeX side directly, rather than being passed to Pygments and then returned. This makes caching more efficient, since code is no longer rehighlighted just because fancyvrb options changed.
- Fixed buffer size error caused by using `cache` with a very large number of files (#61).
- Fixed `autogobble` bug that caused failure under some operating systems.
- Added support for `escapeinside` (requires Pygments 2.0+; #38).

- Fixed issues with XeTeX and caching (#40).
- The `upquote` package now works correctly with single quotes when using Pygments 1.6+ (#34).
- Fixed caching incompatibility with Linux and OS X under xelatex (#18 and #42).
- Fixed `autogobble` incompatibility with Linux and OS X.
- `\mintinline` and derived commands are now robust, via `\newrobustcmd` from `etoolbox`.
- Unused styles are now cleaned up when caching.
- Fixed a bug that could interfere with caching (#24).
- Added `draft` package option (#39). This typesets all code using `fancyvrb`; Pygments is not used. This trades syntax highlighting for maximum speed in compiling.
- Added automatic line breaking with `breaklines` and related options (#1).
- Fixed a bug with boolean options that needed a False argument to cooperate with `\setminted` (#48).

v2.0-alpha3 (2013/12/21)

- Added `autogobble` option. This sends code through Python's `textwrap.dedent()` to remove common leading whitespace.
- Added package option `cachedir`. This allows the directory in which cached content is saved to be specified.
- Added package option `outputdir`. This allows an output directory for temporary files to be specified, so that the package can work with LaTeX's `-output-directory` command-line option.
- The `kvoptions` package is now required. It is needed to process key-value package options, such as the new `cachedir` option.
- Many small improvements, including better handling of paths under Windows and improved key system.

v2.0-alpha2 (2013/08/21)

- `\DeleteFile` now only deletes files if they do indeed exist. This eliminates warning messages due to missing files.
- Fixed a bug in the definition of `\DeleteFile` for non-Windows systems.
- Added support for Pygments option `stripnl`.
- Settings macros that were previously defined globally are now defined locally, so that `\setminted` may be confined by `\begingroup...\endgroup` as expected.

- Macro definitions for a given style are now loaded only once per document, rather than once per command/environment. This works even without caching.
- A custom script/executable may now be substituted for `pygmentize` by redefining `\MintedPygmentize`.

v2.0alpha (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.

1.7 (2011/09/17)

- Options for float placement added [2011/09/12]
- Fixed `tabsize` option [2011/08/30]
- More robust detection of the `-shell-escape` option [2011/01/21]
- Added the `label` option [2011/01/04]
- Installation instructions added [2010/03/16]
- Minimal working example added [2010/03/16]
- Added PHP-specific options [2010/03/14]
- Removed unportable flag from Unix shell command [2010/02/16]

1.6 (2010/01/31)

- Added font-related options [2010/01/27]
- Windows support added [2010/01/27]
- Added command shortcuts [2010/01/22]
- Simpler versioning scheme [2010/01/22]

0.1.5 (2010/01/13)

- Added `fillcolor` option [2010/01/10]
- Added float support [2010/01/10]
- Fixed `firstnumber` option [2010/01/10]
- Removed `caption` option [2010/01/10]

0.0.4 (2010/01/08)

- Initial version [2010/01/08]

9 Implementation

9.1 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{kvoptions}
\RequirePackage{fancyvrb}
\RequirePackage{float}
\RequirePackage{ifthen}
\RequirePackage{calc}
\RequirePackage{ifplatform}
\RequirePackage{pdftexcmds}
\RequirePackage{etoolbox}
\RequirePackage{xstring}
\RequirePackage{lineno}
\RequirePackage{framed}
\IfFileExists{shellesc.sty}
 {\RequirePackage{shellesc}
  \@ifpackagelater{shellesc}{2016/04/29}
   {}
   {\protected\def\ShellEscape{\immediate\write18 }}}
 {\protected\def\ShellEscape{\immediate\write18 }}
```

Make sure that either `color` or `xcolor` is loaded by the beginning of the document.

```
\AtBeginDocument{%
```

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

9.2 Package options

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

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

newfloat Define an option to use newfloat rather than float to create a floated `listing` environment.

```
\DeclareBoolOption{newfloat}
```

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

```
\DeclareBoolOption[true]{cache}
```

\minted@jobname At various points, temporary files and directories will need to be named after the main `.tex` file. The typical way to do this is to use `\jobname`. However, if the file name contains spaces, then `\jobname` will contain the name wrapped in quotes (older versions of MiKTeX replace spaces with asterisks instead, and `XeTeX` apparently allows double quotes within file names, in which case names are wrapped in single quotes). While that is perfectly fine for working with LaTeX internally, it causes problems with `\write18`, since quotes will end up in unwanted locations in shell commands. It would be possible to strip the wrapping quotation marks when they are present, and maintain any spaces in the file name. But it is simplest to create a "sanitized" version of `\jobname` in which spaces and asterisks are replaced by underscores, and double quotes are stripped.

```
\StrSubstitute{\jobname}{ }{_}[\minted@jobname]
\StrSubstitute{\minted@jobname}{*}{_}[\minted@jobname]
\StrSubstitute{\minted@jobname}{"}{}[\minted@jobname]
```

\minted@cachedir Set the directory in which cached content is saved. The default uses a `minted-` prefix followed by the sanitized `\minted@jobname`.

```
\newcommand{\minted@cachedir}{\detokenize{_}minted-\minted@jobname}
\let\minted@cachedir@windows\minted@cachedir
\define@key{minted}{cachedir}{%
  \@namedef{minted@cachedir}{#1}%
  \StrSubstitute{\minted@cachedir}{/}{\@backslashchar}[\minted@cachedir@windows]}
```

`finalizecache` Define an option that switches the naming of cache files from an MD5-based system to a `listing<number>` scheme. Compiling with this option is a prerequisite to turning on `frozencache`.

```
\DeclareBoolOption{finalizecache}
```

`frozencache` Define an option that uses a fixed set of cache files, using `listing<number>` file naming with `\write18` disabled. This is convenient for working with a document in an environment in which `\write18` support is disabled and minted content does not need to be modified.

```
\DeclareBoolOption{frozencache}
```

`\minted@outputdir` The `-output-directory` command-line option for LaTeX causes problems for minted, because the minted temporary files are saved in the output directory, but minted still looks for them in the document root directory. There is no way to access the value of the command-line option. But it is possible to allow the output directory to be specified manually as a package option. A trailing slash is automatically appended to the `outputdir`, so that it may be directly joined to `cachedir`. This may be redundant if the user-supplied value already ends with a slash, but doubled slashes are ignored under *nix and Windows, so it isn't a problem.

```
\let\minted@outputdir\@empty
\let\minted@outputdir@windows\@empty
\define@key{minted}{outputdir}{%
  \@namedef{minted@outputdir}{#1/}%
  \StrSubstitute{\minted@outputdir}{/}%
    {\@backslashchar}[\minted@outputdir@windows]}
```

`kpsewhich` Define an option that invokes `kpsewhich` to locate the files that are to be `pygmentized`. This isn't done by default to avoid the extra overhead, but can be useful with some build tools such as `texi2pdf` that rely on modifying `TEXINPUTS`.

```
\DeclareBoolOption{kpsewhich}
```

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

```
\DeclareBoolOption{langlinenos}
```

`draft` Define an option that allows fancyvrb to do all typesetting directly, without using Pygments. This trades syntax highlighting for speed. Note that in many cases, the difference in performance between caching and draft mode will be minimal. Also note that draft settings may be inherited from the document class.

```
\DeclareBoolOption{draft}
```

final Define a `final` option that is the opposite of `draft`, since many packages do this.

```
\DeclareComplementaryOption{final}{draft}
```

Process package options. Proceed with everything that immediately relies upon them. If PGF/Ti*k*Z externalization is in use, switch on `draft` mode and turn off `cache`. Externalization involves compiling the *entire* document; all parts not related to the current image are “silently thrown away.” minted needs to cooperate with that by not writing any temp files or creating any directories. Two checks are done for externalization. The first, using `\tikzifexternalizing`, works if externalization is set before minted is loaded. The second, using `\tikzexternalrealjob`, works if externalization is set after minted is loaded.

```
\ProcessKeyvalOptions*
\ifthenelse{\boolean{minted@newfloat}}{\RequirePackage{newfloat}}{}
\ifcsname tikzifexternalizing\endcsname
  \tikzifexternalizing{\minted@drafttrue\minted@cachefalse}{}
\else
  \ifcsname tikzexternalrealjob\endcsname
    \minted@drafttrue
    \minted@cachefalse
  \else
  \fi
\fi
\ifthenelse{\boolean{minted@finalizecache}}%
 {\ifthenelse{\boolean{minted@frozencache}}%
   {\PackageError{minted}%
     {Options "finalizecache" and "frozencache" are not compatible}%
     {Options "finalizecache" and "frozencache" are not compatible}}%
   {}}%
 {}
\ifthenelse{\boolean{minted@cache}}%
 {\ifthenelse{\boolean{minted@frozencache}}%
   {}%
   {\AtEndOfPackage{\ProvideDirectory{\minted@outputdir\minted@cachedir}}}}%
 {}
```

9.3 Input, caching, and temp files

\minted@input We need a wrapper for `\input`. In most cases, `\input` failure will be due to attempts to use `\inputminted` with files that don't exist, but we also want to give informative error messages when `outputdir` is needed or incompatible build tools are used.

```
\newcommand{\minted@input}[1]{%
  \IfFileExists{#1}%
   {\input{#1}}%
   {\PackageError{minted}{Missing Pygments output; \string\inputminted\space
```

```
    was^^Jprobably given a file that does not exist--otherwise, you may need
    ^^Jthe outputdir package option, or may be using an incompatible build
    tool\ifwindows,^^Jor may be using the kpsewhich option without having
    PowerShell installed\fi,^^Jor may be using frozencache with a missing file}%
   {This could be caused by using -output-directory or -aux-directory
    ^^Jwithout setting minted's outputdir, or by using a build tool that
    ^^Jchanges paths in ways minted cannot detect\ifwindows, or by using the
    ^^Jkpsewhich option without PowerShell\fi,
    ^^Jor using frozencache with a missing file.}}%
}
```

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

We need a way to track the cache files that are created, and delete those that are not in use. This is accomplished by creating a comma-delimited list of cache files and saving this list to the .aux file so that it may be accessed on subsequent runs. During subsequent runs, this list is compared against the cache files that are actually used, and unused files are deleted. Cache file names are created with MD5 hashes of highlighting settings and file contents, with a .pygtex extension, so they never contain commas. Thus comma-delimiting the list of file names doesn't introduce a potential for errors.

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

```
\newcommand{\minted@cachelist}{}
```

\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 by seeing if the macro exists. The list of cache files must be created with built-in linebreaks, so that when it is written to the .aux file, it won't all be on one line and thereby risk buffer errors.

```
\newcommand{\minted@addcachefile}[1]{%
  \expandafter\long\expandafter\gdef\expandafter\minted@cachelist\expandafter{%
    \minted@cachelist,^^J%
    \space\space#1}%
  \expandafter\gdef\csname minted@cached@#1\endcsname{}%
}
```

\minted@savecachelist 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@savecachelist}{%
  \ifdefempty{\minted@cachelist}{}{%
```

```
    \immediate\write\@mainaux{%
      \string\gdef\string\minted@oldcachelist\string{%
        \minted@cachelist\string}}%
  }%
}
```

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

```
\newcommand{\minted@cleancache}{%
  \ifcsname minted@oldcachelist\endcsname
    \def\do##1{%
      \ifthenelse{\equal{##1}{}}{}{%
        \ifcsname minted@cached@##1\endcsname\else
          \DeleteFile[\minted@outputdir\minted@cachedir]{##1}%
        \fi
      }%
    }%
    \expandafter\docsvlist\expandafter{\minted@oldcachelist}%
  \else
  \fi
}
```

At the end of the document, save the list of cache files and clean the cache. If in draft mode, don't clean up the cache and save the old cache file list for next time. This allows draft mode to be switched on and off without requiring that all highlighted content be regenerated. The saving and cleaning operations may be called without conditionals, since their definitions already contain all necessary checks for their correct operation.

```
\ifthenelse{\boolean{minted@draft}}%
 {\AtEndDocument{%
    \ifcsname minted@oldcachelist\endcsname
      \StrSubstitute{\minted@oldcachelist}{,}{,^^J }[\minted@cachelist]
      \minted@savecachelist
    \fi}}%
 {\ifthenelse{\boolean{minted@frozencache}}%
   {\AtEndDocument{%
      \ifcsname minted@oldcachelist\endcsname
        \StrSubstitute{\minted@oldcachelist}{,}{,^^J }[\minted@cachelist]
        \minted@savecachelist
      \fi}}%
   {\AtEndDocument{%
    \minted@savecachelist
    \minted@cleancache}}}%
```

9.4 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}{}}%
      {\IfFileExists{#2}{\ShellEscape{del #2}}{}}%
      {\IfFileExists{#1/#2}{%
        \StrSubstitute{#1}{/}{\@backslashchar}[\minted@windir]
        \ShellEscape{del \minted@windir\@backslashchar #2}}{}}}
\else
  \providecommand{\DeleteFile}[2][]{%
    \ifthenelse{\equal{#1}{}}%
      {\IfFileExists{#2}{\ShellEscape{rm #2}}{}}%
      {\IfFileExists{#1/#2}{\ShellEscape{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]{%
    \StrSubstitute{#1}{/}{\@backslashchar}[\minted@windir]
    \ShellEscape{if not exist \minted@windir\space mkdir \minted@windir}}
\else
  \newcommand{\ProvideDirectory}[1]{%
    \ShellEscape{mkdir -p #1}}
\fi
```

`\TestAppExists` Determine whether a given application exists.

Usage is a bit roundabout, but has been retained for backward compatibility. At some point, it may be worth replacing this with something using `\@@input"|<command>"`. That would require MiKTeX users to `--enable-pipes`, however, which would make things a little more complicated. If Windows XP compatibility is ever no longer required, the `where` command could be used instead of the approach for Windows.

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{\minted@jobname.aex}
    \ShellEscape{for \string^\@percentchar i in (#1.exe #1.bat #1.cmd)
      do set > \minted@jobname.aex <nul: /p
      x=\string^\@percentchar \string~$PATH:i>> \minted@jobname.aex}
    %$ <- balance syntax highlighting
    \immediate\openin\minted@appexistsfile\minted@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{\minted@jobname.aex}
  \else
```

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

```
    \ShellEscape{which #1 && touch \minted@jobname.aex}
    \IfFileExists{\minted@jobname.aex}
      {\AppExiststrue
        \DeleteFile{\minted@jobname.aex}}
      {\AppExistsfalse}
  \fi
}
```

9.5 Option processing

Option processing is somewhat involved, because we want to be able to define options at various levels of hierarchy: individual command/environment, language, global (document). And once those options are defined, we need to go through the hierarchy in a defined order of precedence to determine which option to apply. As if that wasn't complicated enough, some options need to be sent to Pygments, some need to be sent to fancyvrb, and some need to be processed within minted itself.

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

`\minted@optlistcl@lang` may be `\let` to this macro as convenient; otherwise, the general language macro merely serves as a placeholder.

The global- and language-level lists also have an `inline` (`i`) variant. This allows different settings to be applied in inline settings. An inline variant is not needed at the command/environment level, since at that level settings would not be present unless they were supposed to be applied.

`\minted@optlistcl@g`

```
\newcommand{\minted@optlistcl@g}{}
```

`\minted@optlistcl@g@i`

```
\newcommand{\minted@optlistcl@g@i}{}
```

`\minted@lang`

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

`\minted@optlistcl@lang`

```
\newcommand{\minted@optlistcl@lang}{}
```

`\minted@optlistcl@lang@i`

```
\newcommand{\minted@optlistcl@lang@i}{}
```

`\minted@optlistcl@cmd`

```
\newcommand{\minted@optlistcl@cmd}{}
```

We also need macros for storing lists of options that will later be passed to fancyvrb (`optlistfv`). As before, these exist at the global (`fv@g`), language (`fv@lang`), and command or environment (`fv@cmd`) levels. Pygments accepts fancyvrb options, but in almost all cases, these options may be applied via `\fvset` rather than via running Pygments. This is significantly more efficient when caching is turned on, since it allows formatting changes to be applied without having to re-highlight the code.

`\minted@optlistfv@g`

```
\newcommand{\minted@optlistfv@g}{}
```

`\minted@optlistfv@g@i`

```
\newcommand{\minted@optlistfv@g@i}{}
```

\minted@optlistfv@lang

```
\newcommand{\minted@optlistfv@lang}{}
```

\minted@optlistfv@lang@i

```
\newcommand{\minted@optlistfv@lang@i}{}
```

\minted@optlistfv@cmd

```
\newcommand{\minted@optlistfv@cmd}{}
```

\minted@configlang We need a way to check whether a language has had all its option list macros created. This generally occurs in a context where `\minted@lang` needs to be set. So we create a macro that does both at once. If the language list macros do not exist, we create them globally to simplify future operations.

```
\newcommand{\minted@configlang}[1]{%
  \def\minted@lang{#1}%
  \ifcsname minted@optlistcl@lang\minted@lang\endcsname\else
    \expandafter\gdef\csname minted@optlistcl@lang\minted@lang\endcsname{}%
  \fi
  \ifcsname minted@optlistcl@lang\minted@lang @i\endcsname\else
    \expandafter\gdef\csname minted@optlistcl@lang\minted@lang @i\endcsname{}%
  \fi
  \ifcsname minted@optlistfv@lang\minted@lang\endcsname\else
    \expandafter\gdef\csname minted@optlistfv@lang\minted@lang\endcsname{}%
  \fi
  \ifcsname minted@optlistfv@lang\minted@lang @i\endcsname\else
    \expandafter\gdef\csname minted@optlistfv@lang\minted@lang @i\endcsname{}%
  \fi
}
```

We need a way to define options in bulk at the global, language, and command levels. How this is done will depend on the type of option. The keys created are grouped by level: `minted@opt@g`, `minted@opt@lang`, and `minted@opt@cmd`, plus inline variants. The language-level key groupings use `\minted@lang` internally, so we don't need to duplicate the internals for different languages. The key groupings are independent of whether a given option relates to Pygments, fancyvrb, etc. Organization by level is the only thing that is important here, since keys are applied in a hierarchical fashion. Key values are stored in macros of the form `\minted@opt@`⟨*level*⟩`:`⟨*key*⟩, so that they may be retrieved later. In practice, these key macros will generally not be used directly (hence the colon in the name). Rather, the hierarchy of macros will be traversed until an existing macro is found.

\minted@def@optcl Define a generic option that will be passed to the command line. Options are given in a `{key}{value}` format that is transformed into `key=value` and then passed to pygmentize. This allows value to be easily stored in a separate macro for later access. This is useful, for example, in separately accessing the value of encoding for performing autogobble.

If a key option is specified without =value, the default is assumed. Options are automatically created at all levels.

Options are added to the option lists in such a way that they will be detokenized. This is necessary since they will ultimately be used in \write18.

```
\newcommand{\minted@addto@optlistcl}[2]{%
  \expandafter\def\expandafter#1\expandafter{#1%
    \detokenize{#2}\space}}
\newcommand{\minted@addto@optlistcl@lang}[2]{%
  \expandafter\let\expandafter\minted@tmp\csname #1\endcsname
  \expandafter\def\expandafter\minted@tmp\expandafter{\minted@tmp%
    \detokenize{#2}\space}%
  \expandafter\let\csname #1\endcsname\minted@tmp}
\newcommand{\minted@def@optcl}[4][]{%
  \ifthenelse{\equal{#1}{}}%
    {\define@key{minted@opt@g}{#2}{%
        \minted@addto@optlistcl{\minted@optlistcl@g}{#3=#4}%
        \@namedef{minted@opt@g:#2}{#4}}%
      \define@key{minted@opt@g@i}{#2}{%
        \minted@addto@optlistcl{\minted@optlistcl@g@i}{#3=#4}%
        \@namedef{minted@opt@g@i:#2}{#4}}%
      \define@key{minted@opt@lang}{#2}{%
        \minted@addto@optlistcl@lang{minted@optlistcl@lang\minted@lang}{#3=#4}%
        \@namedef{minted@opt@lang\minted@lang:#2}{#4}}%
      \define@key{minted@opt@lang@i}{#2}{%
        \minted@addto@optlistcl@lang{%
          minted@optlistcl@lang\minted@lang @i}{#3=#4}%
        \@namedef{minted@opt@lang\minted@lang @i:#2}{#4}}%
      \define@key{minted@opt@cmd}{#2}{%
        \minted@addto@optlistcl{\minted@optlistcl@cmd}{#3=#4}%
        \@namedef{minted@opt@cmd:#2}{#4}}}%
    {\define@key{minted@opt@g}{#2}[#1]{%
        \minted@addto@optlistcl{\minted@optlistcl@g}{#3=#4}%
        \@namedef{minted@opt@g:#2}{#4}}%
      \define@key{minted@opt@g@i}{#2}[#1]{%
        \minted@addto@optlistcl{\minted@optlistcl@g@i}{#3=#4}%
        \@namedef{minted@opt@g@i:#2}{#4}}%
      \define@key{minted@opt@lang}{#2}[#1]{%
        \minted@addto@optlistcl@lang{minted@optlistcl@lang\minted@lang}{#3=#4}%
        \@namedef{minted@opt@lang\minted@lang:#2}{#4}}%
      \define@key{minted@opt@lang@i}{#2}[#1]{%
        \minted@addto@optlistcl@lang{%
          minted@optlistcl@lang\minted@lang @i}{#3=#4}%
        \@namedef{minted@opt@lang\minted@lang @i:#2}{#4}}%
      \define@key{minted@opt@cmd}{#2}[#1]{%
```

```
        \minted@addto@optlistcl{\minted@optlistcl@cmd}{#3=#4}%
        \@namedef{minted@opt@cmd:#2}{#4}}}%
}
```

This covers the typical options that must be passed to Pygments. But some, particularly `escapeinside`, need more work. Since their arguments may contain escaped characters, expansion rather than detokenization is needed. Getting expansion to work as desired in a `\write18` context requires the redefinition of some characters.

`\minted@escchars` We need to define versions of common escaped characters that will work correctly under expansion for use in `\write18`.

```
\edef\minted@hashchar{\string#}
\edef\minted@dollarchar{\string$}
\edef\minted@ampchar{\string&}
\edef\minted@underscorechar{\string_}
\edef\minted@tildechar{\string~}
\edef\minted@leftsquarebracket{\string[}
\edef\minted@rightsquarebracket{\string]}
\newcommand{\minted@escchars}{%
  \let\#\minted@hashchar
  \let\%\@percentchar
  \let\{\@charlb
  \let\}\@charrb
  \let\$\minted@dollarchar
  \let\&\minted@ampchar
  \let\_\minted@underscorechar
  \let\\\@backslashchar
  \let~\minted@tildechar
  \let\~\minted@tildechar
  \let\[\minted@leftsquarebracket
  \let\]\minted@rightsquarebracket
} %$ <- highlighting
```

`\minted@def@optcl@e` Now to define options that are expanded.

```
\newcommand{\minted@addto@optlistcl@e}[2]{%
  \begingroup
  \minted@escchars
  \xdef\minted@xtmp{#2}%
  \endgroup
  \expandafter\minted@addto@optlistcl@e@i\expandafter{\minted@xtmp}{#1}}
\def\minted@addto@optlistcl@e@i#1#2{%
  \expandafter\def\expandafter#2\expandafter{#2#1\space}}
\newcommand{\minted@addto@optlistcl@lang@e}[2]{%
  \begingroup
  \minted@escchars
  \xdef\minted@xtmp{#2}%
  \endgroup
  \expandafter\minted@addto@optlistcl@lang@e@i\expandafter{\minted@xtmp}{#1}}
\def\minted@addto@optlistcl@lang@e@i#1#2{%
  \expandafter\let\expandafter\minted@tmp\csname #2\endcsname
  \expandafter\def\expandafter\minted@tmp\expandafter{\minted@tmp#1\space}%
  \expandafter\let\csname #2\endcsname\minted@tmp}
\newcommand{\minted@def@optcl@e}[4][]{%
  \ifthenelse{\equal{#1}{}}%
    {\define@key{minted@opt@g}{#2}{%
        \minted@addto@optlistcl@e{\minted@optlistcl@g}{#3=#4}%
        \@namedef{minted@opt@g:#2}{#4}}%
      \define@key{minted@opt@g@i}{#2}{%
        \minted@addto@optlistcl@e{\minted@optlistcl@g@i}{#3=#4}%
        \@namedef{minted@opt@g@i:#2}{#4}}%
      \define@key{minted@opt@lang}{#2}{%
        \minted@addto@optlistcl@lang@e{minted@optlistcl@lang\minted@lang}{#3=#4}%
        \@namedef{minted@opt@lang\minted@lang:#2}{#4}}%
      \define@key{minted@opt@lang@i}{#2}{%
        \minted@addto@optlistcl@lang@e{%
          minted@optlistcl@lang\minted@lang @i}{#3=#4}%
        \@namedef{minted@opt@lang\minted@lang @i:#2}{#4}}%
      \define@key{minted@opt@cmd}{#2}{%
        \minted@addto@optlistcl@e{\minted@optlistcl@cmd}{#3=#4}%
        \@namedef{minted@opt@cmd:#2}{#4}}}%
    {\define@key{minted@opt@g}{#2}[#1]{%
        \minted@addto@optlistcl@e{\minted@optlistcl@g}{#3=#4}%
        \@namedef{minted@opt@g:#2}{#4}}%
      \define@key{minted@opt@g@i}{#2}[#1]{%
        \minted@addto@optlistcl@e{\minted@optlistcl@g@i}{#3=#4}%
        \@namedef{minted@opt@g@i:#2}{#4}}%
      \define@key{minted@opt@lang}{#2}[#1]{%
        \minted@addto@optlistcl@lang@e{minted@optlistcl@lang\minted@lang}{#3=#4}%
        \@namedef{minted@opt@lang\minted@lang:#2}{#4}}%
      \define@key{minted@opt@lang@i}{#2}[#1]{%
        \minted@addto@optlistcl@lang@e{%
          minted@optlistcl@lang\minted@lang @i}{#3=#4}%
        \@namedef{minted@opt@lang\minted@lang @i:#2}{#4}}%
      \define@key{minted@opt@cmd}{#2}[#1]{%
        \minted@addto@optlistcl@e{\minted@optlistcl@cmd}{#3=#4}%
        \@namedef{minted@opt@cmd:#2}{#4}}}%
}
```

`\minted@def@optcl@switch` Define a switch or boolean option that is passed to Pygments, which is `true` when no value is specified.

```
\newcommand{\minted@def@optcl@switch}[2]{%
  \define@booleankey{minted@opt@g}{#1}%
    {\minted@addto@optlistcl{\minted@optlistcl@g}{#2=True}%
      \@namedef{minted@opt@g:#1}{true}}
    {\minted@addto@optlistcl{\minted@optlistcl@g}{#2=False}%
```

```
      \@namedef{minted@opt@g:#1}{false}}
  \define@booleankey{minted@opt@g@i}{#1}%
    {\minted@addto@optlistcl{\minted@optlistcl@g@i}{#2=True}%
      \@namedef{minted@opt@g@i:#1}{true}}
    {\minted@addto@optlistcl{\minted@optlistcl@g@i}{#2=False}%
      \@namedef{minted@opt@g@i:#1}{false}}
  \define@booleankey{minted@opt@lang}{#1}%
    {\minted@addto@optlistcl@lang{minted@optlistcl@lang\minted@lang}{#2=True}%
      \@namedef{minted@opt@lang\minted@lang:#1}{true}}
    {\minted@addto@optlistcl@lang{minted@optlistcl@lang\minted@lang}{#2=False}%
      \@namedef{minted@opt@lang\minted@lang:#1}{false}}
  \define@booleankey{minted@opt@lang@i}{#1}%
    {\minted@addto@optlistcl@lang{minted@optlistcl@lang\minted@lang @i}{#2=True}%
      \@namedef{minted@opt@lang\minted@lang @i:#1}{true}}
    {\minted@addto@optlistcl@lang{minted@optlistcl@lang\minted@lang @i}{#2=False}%
      \@namedef{minted@opt@lang\minted@lang @i:#1}{false}}
  \define@booleankey{minted@opt@cmd}{#1}%
      {\minted@addto@optlistcl{\minted@optlistcl@cmd}{#2=True}%
        \@namedef{minted@opt@cmd:#1}{true}}
      {\minted@addto@optlistcl{\minted@optlistcl@cmd}{#2=False}%
        \@namedef{minted@opt@cmd:#1}{false}}
}
```

Now that all the machinery for Pygments options is in place, we can move on to fancyvrb options.

\minted@def@optfv

Define fancyvrb options.

```
\newcommand{\minted@def@optfv}[1]{%
  \define@key{minted@opt@g}{#1}{%
    \expandafter\def\expandafter\minted@optlistfv@g\expandafter{%
      \minted@optlistfv@g#1=##1,}%
    \@namedef{minted@opt@g:#1}{##1}}
  \define@key{minted@opt@g@i}{#1}{%
    \expandafter\def\expandafter\minted@optlistfv@g@i\expandafter{%
      \minted@optlistfv@g@i#1=##1,}%
    \@namedef{minted@opt@g@i:#1}{##1}}
  \define@key{minted@opt@lang}{#1}{%
    \expandafter\let\expandafter\minted@tmp%
      \csname minted@optlistfv@lang\minted@lang\endcsname
    \expandafter\def\expandafter\minted@tmp\expandafter{%
      \minted@tmp#1=##1,}%
    \expandafter\let\csname minted@optlistfv@lang\minted@lang\endcsname%
      \minted@tmp
    \@namedef{minted@opt@lang\minted@lang:#1}{##1}}
  \define@key{minted@opt@lang@i}{#1}{%
    \expandafter\let\expandafter\minted@tmp%
      \csname minted@optlistfv@lang\minted@lang @i\endcsname
    \expandafter\def\expandafter\minted@tmp\expandafter{%
```

```
      \minted@tmp#1=##1,}%
    \expandafter\let\csname minted@optlistfv@lang\minted@lang @i\endcsname%
      \minted@tmp
    \@namedef{minted@opt@lang\minted@lang @i:#1}{##1}}
  \define@key{minted@opt@cmd}{#1}{%
    \expandafter\def\expandafter\minted@optlistfv@cmd\expandafter{%
      \minted@optlistfv@cmd#1=##1,}%
    \@namedef{minted@opt@cmd:#1}{##1}}
}
```

\minted@def@optfv@switch Define fancyvrb boolean options.

```
\newcommand{\minted@def@optfv@switch}[1]{%
  \define@booleankey{minted@opt@g}{#1}%
    {\expandafter\def\expandafter\minted@optlistfv@g\expandafter{%
      \minted@optlistfv@g#1=true,}%
     \@namedef{minted@opt@g:#1}{true}}%
    {\expandafter\def\expandafter\minted@optlistfv@g\expandafter{%
      \minted@optlistfv@g#1=false,}%
     \@namedef{minted@opt@g:#1}{false}}%
  \define@booleankey{minted@opt@g@i}{#1}%
    {\expandafter\def\expandafter\minted@optlistfv@g@i\expandafter{%
      \minted@optlistfv@g@i#1=true,}%
     \@namedef{minted@opt@g@i:#1}{true}}%
    {\expandafter\def\expandafter\minted@optlistfv@g@i\expandafter{%
      \minted@optlistfv@g@i#1=false,}%
     \@namedef{minted@opt@g@i:#1}{false}}%
  \define@booleankey{minted@opt@lang}{#1}%
    {\expandafter\let\expandafter\minted@tmp%
        \csname minted@optlistfv@lang\minted@lang\endcsname
      \expandafter\def\expandafter\minted@tmp\expandafter{%
        \minted@tmp#1=true,}%
      \expandafter\let\csname minted@optlistfv@lang\minted@lang\endcsname%
        \minted@tmp
     \@namedef{minted@opt@lang\minted@lang:#1}{true}}%
    {\expandafter\let\expandafter\minted@tmp%
        \csname minted@optlistfv@lang\minted@lang\endcsname
      \expandafter\def\expandafter\minted@tmp\expandafter{%
        \minted@tmp#1=false,}%
      \expandafter\let\csname minted@optlistfv@lang\minted@lang\endcsname%
        \minted@tmp
     \@namedef{minted@opt@lang\minted@lang:#1}{false}}%
  \define@booleankey{minted@opt@lang@i}{#1}%
    {\expandafter\let\expandafter\minted@tmp%
        \csname minted@optlistfv@lang\minted@lang @i\endcsname
      \expandafter\def\expandafter\minted@tmp\expandafter{%
        \minted@tmp#1=true,}%
      \expandafter\let\csname minted@optlistfv@lang\minted@lang @i\endcsname%
        \minted@tmp
     \@namedef{minted@opt@lang\minted@lang @i:#1}{true}}%
    {\expandafter\let\expandafter\minted@tmp%
        \csname minted@optlistfv@lang\minted@lang @i\endcsname
      \expandafter\def\expandafter\minted@tmp\expandafter{%
        \minted@tmp#1=false,}%
      \expandafter\let\csname minted@optlistfv@lang\minted@lang @i\endcsname%
        \minted@tmp
     \@namedef{minted@opt@lang\minted@lang @i:#1}{false}}%
  \define@booleankey{minted@opt@cmd}{#1}%
    {\expandafter\def\expandafter\minted@optlistfv@cmd\expandafter{%
      \minted@optlistfv@cmd#1=true,}%
     \@namedef{minted@opt@cmd:#1}{true}}%
    {\expandafter\def\expandafter\minted@optlistfv@cmd\expandafter{%
      \minted@optlistfv@cmd#1=false,}%
     \@namedef{minted@opt@cmd:#1}{false}}%
}
```

`minted@isinline` In resolving value precedence when actually using values, we need a way to determine whether we are in an inline context. This is accomplished via a boolean that is set at the beginning of inline commands.

```
\newboolean{minted@isinline}
```

`\minted@fvset` We will need a way to actually use the lists of stored fancyvrb options later on.

```
\newcommand{\minted@fvset}{%
  \expandafter\fvset\expandafter{\minted@optlistfv@g}%
  \expandafter\let\expandafter\minted@tmp%
    \csname minted@optlistfv@lang\minted@lang\endcsname
  \expandafter\fvset\expandafter{\minted@tmp}%
  \ifthenelse{\boolean{minted@isinline}}%
   {\expandafter\fvset\expandafter{\minted@optlistfv@g@i}%
    \expandafter\let\expandafter\minted@tmp%
      \csname minted@optlistfv@lang\minted@lang @i\endcsname
    \expandafter\fvset\expandafter{\minted@tmp}}%
   {}%
  \expandafter\fvset\expandafter{\minted@optlistfv@cmd}%
}
```

We need a way to define minted-specific options at multiple levels of hierarchy, as well as a way to retrieve these options. As with previous types of options, values are stored in macros of the form `\minted@opt@`⟨*level*⟩`:`⟨*key*⟩, since they are not meant to be accessed directly.

The order of precedence is `cmd`, `lang@i`, `g@i`, `lang`, `g`. A value specified at the command or environment level should override other settings. In its absence, a value specified for an inline command should override other settings, if we are indeed in an inline context. Otherwise, language settings take precedence over global settings.

Before actually creating the option-definition macro, we need a few helper macros.

`\minted@def@opt` Finally, on to the actual option definitions for minted-specific options.

Usage: `\minted@def@opt[`⟨*initial global value*⟩`]{`⟨*key name*⟩`}`

```
\newcommand{\minted@def@opt}[2][]{%
  \define@key{minted@opt@g}{#2}{%
    \@namedef{minted@opt@g:#2}{##1}}
  \define@key{minted@opt@g@i}{#2}{%
    \@namedef{minted@opt@g@i:#2}{##1}}
  \define@key{minted@opt@lang}{#2}{%
    \@namedef{minted@opt@lang\minted@lang:#2}{##1}}
  \define@key{minted@opt@lang@i}{#2}{%
    \@namedef{minted@opt@lang\minted@lang @i:#2}{##1}}
  \define@key{minted@opt@cmd}{#2}{%
    \@namedef{minted@opt@cmd:#2}{##1}}
}
```

`\minted@def@opt@style` Define an option for styles. These are defined independently because styles need slightly different handling. It is conventient to create style macros when styles are set. Otherwise, it would be necessary to check for the existence of style macros at the beginning of every command or environment.

```
\newcommand{\minted@def@opt@style}{%
  \define@key{minted@opt@g}{style}{%
    \minted@checkstyle{##1}%
    \@namedef{minted@opt@g:style}{##1}}%
  \define@key{minted@opt@g@i}{style}{%
    \minted@checkstyle{##1}%
    \@namedef{minted@opt@g@i:style}{##1}}%
  \define@key{minted@opt@lang}{style}{%
    \minted@checkstyle{##1}%
    \@namedef{minted@opt@lang\minted@lang:style}{##1}}%
  \define@key{minted@opt@lang@i}{style}{%
    \minted@checkstyle{##1}%
    \@namedef{minted@opt@lang\minted@lang @i:style}{##1}}%
  \define@key{minted@opt@cmd}{style}{%
    \minted@checkstyle{##1}%
    \@namedef{minted@opt@cmd:style}{##1}}%
}
```

`\minted@checkstyle` Make sure that style macros exist.

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. There is also the very unorthodox `\let\def\gdef` to make sure that macros are defined globally. The catcodes for - and _ must be changed during macro definition to accomodate style names like `paraiso-light`, `paraiso-dark`, and `algol_nu`.

If a style is not given, then revert to the `default` style, but create macros with prefix `PYG`, and create `default-pyg-prefix.pygstyle` if caching is on. This allows a graceful fallback in the event that style is empty. It is also purposefully used to create a complete set of macros with prefix `PYG`, so that the symbol macros may be used, as described next.

The typical style macros created by `\minted@checkstyle`, which are of the form `\PYG<style>`, are used indirectly. All code is highlighted with `commandprefix=PYG`, so that it uses `\PYG`. Then `\PYG` is `\let` to `\PYG<style>` as appropriate. This way, code need not be highlighted again when the style is changed. This has the disadvantage that none of the `\PYG<symbol>` macros will be defined; rather, only `\PYG<style><symbol>` macros will be defined. It would be possible to `\let` `\PYG<symbol>` to `\PYG<style><symbol>`, but it is simpler to define a complete set of symbol macros using the `PYG` prefix, so that all symbol macros will be defined by default.[6]

```
\newcommand{\minted@checkstyle}[1]{%
  \ifcsname minted@styleloaded@\ifstrempty{#1}{default-pyg-prefix}{#1}\endcsname\el
    \expandafter\gdef%
      \csname minted@styleloaded@\ifstrempty{#1}{default-pyg-prefix}{#1}\endcsname{
    \ifthenelse{\boolean{minted@cache}}%
     {\IfFileExists
       {\minted@outputdir\minted@cachedir/\ifstrempty{#1}{default-pyg-prefix}{#1}.py
       {}%
       {%
        \ifthenelse{\boolean{minted@frozencache}}%
         {\PackageError{minted}%
           {Missing style definition for #1 with frozencache}%
           {Missing style definition for #1 with frozencache}}%
         {\ifwindows
             \ShellEscape{%
               \MintedPygmentize\space -S \ifstrempty{#1}{default}{#1} -f latex
               -P commandprefix=PYG#1
               > \minted@outputdir@windows\minted@cachedir@windows\@backslashchar%
                    \ifstrempty{#1}{default-pyg-prefix}{#1}.pygstyle}%
          \else
             \ShellEscape{%
               \MintedPygmentize\space -S \ifstrempty{#1}{default}{#1} -f latex
               -P commandprefix=PYG#1
               > \minted@outputdir\minted@cachedir/%
                    \ifstrempty{#1}{default-pyg-prefix}{#1}.pygstyle}%
```

[6]It would be possible to hard-code the symbol macros in minted itself, but that would have the disadvantage of tying minted more closely to a particular version of Pygments. Similarly, `\let`ing symbol macros assumes a complete, fixed list of symbol macros. The current approach is harder to break than these alternatives; the worst-case scenario should be needing to purge the cache, rather than dealing with an undefined macro.

```
          \fi}%
        }%
        \begingroup
        \let\def\gdef
        \catcode`\_=11
        \catcode`\-=11
        \endlinechar=-1\relax
        \minted@input{%
          \minted@outputdir\minted@cachedir/\ifstrempty{#1}{default-pyg-prefix}{#1}
        \endgroup
        \minted@addcachefile{\ifstrempty{#1}{default-pyg-prefix}{#1}.pygstyle}}%
     {%
        \ifwindows
          \ShellEscape{%
            \MintedPygmentize\space -S \ifstrempty{#1}{default}{#1} -f latex
            -P commandprefix=PYG#1 > \minted@outputdir@windows\minted@jobname.out.py
        \else
          \ShellEscape{%
            \MintedPygmentize\space -S \ifstrempty{#1}{default}{#1} -f latex
            -P commandprefix=PYG#1 > \minted@outputdir\minted@jobname.out.pyg}%
        \fi
        \begingroup
        \let\def\gdef
        \catcode`\_=11
        \catcode`\-=11
        \endlinechar=-1\relax
        \minted@input{\minted@outputdir\minted@jobname.out.pyg}%
        \endgroup}%
  \fi
}
\ifthenelse{\boolean{minted@draft}}{\renewcommand{\minted@checkstyle}[1]{}}{}
```

At the beginning of the document, create the symbol macros with `PYG` prefix and generate the `default` style. This must wait until `\AtBeginDocument`, because the existence of `pygmentize` isn't tested and may not be final until `\AtEndPreamble`.

```
\AtBeginDocument{\minted@checkstyle{}\setminted{style=default}}
```

`\minted@patch@PYGZsq` Patch the Pygments single quote macro for upquote. The single quote macro from Pygments 1.6+ needs to be patched if the upquote package is in use. The conditionals for the patch definition are borrowed from upquote. Patching is done `\AtBeginDocument`, after the macros will have been created. Patching is only attempted if the macro exists, so that there is a graceful fallback in the event of a custom Pygments stylesheet.

```
\newcommand{\minted@patch@PYGZsq}{%
  \ifcsname PYGZsq\endcsname
    \ifx\upquote@cmtt\minted@undefined\else
      \ifx\encodingdefault\upquote@OTone
```

```
        \ifx\ttdefault\upquote@cmtt
          \expandafter\ifdefstring\expandafter{\csname PYGZsq\endcsname}{\char`\'}%
          {\expandafter\gdef\csname PYGZsq\endcsname{\char13 }}{}%
        \else
          \expandafter\ifdefstring\expandafter{\csname PYGZsq\endcsname}{\char`\'}%
          {\expandafter\gdef\csname PYGZsq\endcsname{\textquotesingle}}{}%
        \fi
      \else
        \expandafter\ifdefstring\expandafter{\csname PYGZsq\endcsname}{\char`\'}%
        {\expandafter\gdef\csname PYGZsq\endcsname{\textquotesingle}}{}%
      \fi
    \fi
  \fi
}
\ifthenelse{\boolean{minted@draft}}{}{\AtBeginDocument{\minted@patch@PYGZsq}}
```

`\minted@def@opt@switch` And we need a switch version.

It would be possible to create a special version of `\minted@get@opt` to work with these, but that would be redundant. During the key processing, any values other than `true` and `false` are filtered out. So when using `\minted@get@opt` later, we know that that part has already been taken care of, and we can just use something like `\ifthenelse{\equal{\minted@get@opt{<opt>}{<default>}}{true}}{...}{...}`. Of course, there is the possibility that a default value has not been set, but `\minted@def@opt@switch` sets a global default of `false` to avoid this. And as usual, Pygments values shouldn't be used without considering whether `\minted@get@opt` needs a fallback value.

```
\newcommand{\minted@def@opt@switch}[2][false]{%
  \define@booleankey{minted@opt@g}{#2}%
    {\@namedef{minted@opt@g:#2}{true}}%
    {\@namedef{minted@opt@g:#2}{false}}
  \define@booleankey{minted@opt@g@i}{#2}%
    {\@namedef{minted@opt@g@i:#2}{true}}%
    {\@namedef{minted@opt@g@i:#2}{false}}
  \define@booleankey{minted@opt@lang}{#2}%
    {\@namedef{minted@opt@lang\minted@lang:#2}{true}}%
    {\@namedef{minted@opt@lang\minted@lang:#2}{false}}
  \define@booleankey{minted@opt@lang@i}{#2}%
    {\@namedef{minted@opt@lang\minted@lang @i:#2}{true}}%
    {\@namedef{minted@opt@lang\minted@lang @i:#2}{false}}
  \define@booleankey{minted@opt@cmd}{#2}%
    {\@namedef{minted@opt@cmd:#2}{true}}%
    {\@namedef{minted@opt@cmd:#2}{false}}%
  \@namedef{minted@opt@g:#2}{#1}%
}
```

`\minted@get@opt` We need a way to traverse the hierarchy of values for a given key and return the current value that has precedence. In doing this, we need to specify a default value to use if no value is found. When working with minted-specific values, there should generally be a default value; in those cases, an empty default may be supplied. But the macro should also work with Pygments settings, which are stored in macros of the same form and will sometimes need to be accessed (for example, `encoding`). In the Pygments case, there may very well be no default values on the LaTeX side, because we are falling back on Pygments' own built-in defaults. There is no need to duplicate those when very few Pygments values are ever needed; it is simpler to specify the default fallback when accessing the macro value.

From a programming perspective, the default argument value needs to be mandatory, so that `\minted@get@opt` can be fully expandable. This significantly simplifies accessing options.

```
\def\minted@get@opt#1#2{%
  \ifcsname minted@opt@cmd:#1\endcsname
    \csname minted@opt@cmd:#1\endcsname
  \else
    \ifminted@isinline
      \ifcsname minted@opt@lang\minted@lang @i:#1\endcsname
        \csname minted@opt@lang\minted@lang @i:#1\endcsname
      \else
        \ifcsname minted@opt@g@i:#1\endcsname
          \csname minted@opt@g@i:#1\endcsname
        \else
          \ifcsname minted@opt@lang\minted@lang:#1\endcsname
            \csname minted@opt@lang\minted@lang:#1\endcsname
          \else
            \ifcsname minted@opt@g:#1\endcsname
              \csname minted@opt@g:#1\endcsname
            \else
              #2%
            \fi
          \fi
        \fi
      \fi
    \else
      \ifcsname minted@opt@lang\minted@lang:#1\endcsname
        \csname minted@opt@lang\minted@lang:#1\endcsname
      \else
        \ifcsname minted@opt@g:#1\endcsname
          \csname minted@opt@g:#1\endcsname
        \else
          #2%
        \fi
      \fi
    \fi
  \fi
}%
```

Actual option definitions. Some of these must be defined conditionally depending on whether we are in `draft` mode; in `draft` mode, we need to emulate Pygments functionality with LaTeX, particularly with fancyvrb, when possible. For example, gobbling must be performed by Pygments when `draft` is off, but when `draft` is on, fancyvrb can perform gobbling.

Lexers.

```
\minted@def@optcl{encoding}{-P encoding}{#1}
\minted@def@optcl{outencoding}{-P outencoding}{#1}
\minted@def@optcl@e{escapeinside}{-P "escapeinside}{#1"}
\minted@def@optcl@switch{stripnl}{-P stripnl}
\minted@def@optcl@switch{stripall}{-P stripall}
% Python console
\minted@def@optcl@switch{python3}{-P python3}
% PHP
\minted@def@optcl@switch{funcnamehighlighting}{-P funcnamehighlighting}
\minted@def@optcl@switch{startinline}{-P startinline}
```

Filters.

```
\ifthenelse{\boolean{minted@draft}}%
  {\minted@def@optfv{gobble}}%
  {\minted@def@optcl{gobble}{-F gobble:n}{#1}}
\minted@def@optcl{codetagify}{-F codetagify:codetags}{#1}
\minted@def@optcl{keywordcase}{-F keywordcase:case}{#1}
```

LaTeX formatter.

```
\minted@def@optcl@switch{texcl}{-P texcomments}
\minted@def@optcl@switch{texcomments}{-P texcomments}
\minted@def@optcl@switch{mathescape}{-P mathescape}
\minted@def@optfv@switch{linenos}
\minted@def@opt@style
```

fancyvrb options.

```
\minted@def@optfv{frame}
\minted@def@optfv{framesep}
\minted@def@optfv{framerule}
\minted@def@optfv{rulecolor}
\minted@def@optfv{numbersep}
\minted@def@optfv{numbers}
\minted@def@optfv{firstnumber}
\minted@def@optfv{stepnumber}
\minted@def@optfv{firstline}
\minted@def@optfv{lastline}
\minted@def@optfv{baselinestretch}
\minted@def@optfv{xleftmargin}
\minted@def@optfv{xrightmargin}
\minted@def@optfv{fillcolor}
```

```
\minted@def@optfv{tabsize}
\minted@def@optfv{fontfamily}
\minted@def@optfv{fontsize}
\minted@def@optfv{fontshape}
\minted@def@optfv{fontseries}
\minted@def@optfv{formatcom}
\minted@def@optfv{label}
\minted@def@optfv{labelposition}
\minted@def@optfv@switch{numberblanklines}
\minted@def@optfv@switch{showspaces}
\minted@def@optfv@switch{resetmargins}
\minted@def@optfv@switch{samepage}
\minted@def@optfv@switch{showtabs}
\minted@def@optfv@switch{obeytabs}
% The following are patches currently added onto fancyvrb
\minted@def@optfv@switch{breaklines}
\minted@def@optfv{breakindent}
\minted@def@optfv@switch{breakautoindent}
\minted@def@optfv{breaksymbol}
\minted@def@optfv{breaksymbolsep}
\minted@def@optfv{breaksymbolindent}
\minted@def@optfv{breaksymbolleft}
\minted@def@optfv{breaksymbolsepleft}
\minted@def@optfv{breaksymbolindentleft}
\minted@def@optfv{breaksymbolright}
\minted@def@optfv{breaksymbolsepright}
\minted@def@optfv{breaksymbolindentright}
\minted@def@optfv{breakbefore}
\minted@def@optfv{breakbeforesymbolpre}
\minted@def@optfv{breakbeforesymbolpost}
\minted@def@optfv@switch{breakbeforegroup}
\minted@def@optfv{breakafter}
\minted@def@optfv@switch{breakaftergroup}
\minted@def@optfv{breakaftersymbolpre}
\minted@def@optfv{breakaftersymbolpost}
\minted@def@optfv@switch{breakanywhere}
\minted@def@optfv{breakanywheresymbolpre}
\minted@def@optfv{breakanywheresymbolpost}
```

Finally, options specific to minted.

An option to force `breaklines` to work at the Pygments token level, rather than at the character level. This is useful in keeping things like strings from being split between lines.

```
\minted@def@opt@switch{breakbytoken}
\minted@def@opt@switch{breakbytokenanywhere}
```

`bgcolor`. The original, `minipage`- and `\colorbox`-based solution was replaced with a `framed`-based solution in version 2.2. A dedicated framing package will often be preferable.

```
\minted@def@opt{bgcolor}
```

Autogobble. We create an option that governs when Python's `textwrap.dedent()` is used to autogobble code.

```
\minted@def@opt@switch{autogobble}
```

`\minted@encoding` When working with encoding, we will need access to the current encoding. That may be done via `\minted@get@opt`, but it is more convenient to go ahead and define a shortcut with an appropriate default

```
\newcommand{\minted@encoding}{\minted@get@opt{encoding}{UTF8}}
```

9.6 Additions to fancyvrb

The following code adds automatic line breaking functionality to fancyvrb's `Verbatim` environment. The code is intentionally written as an extension to fancyvrb, rather than as part of minted. Once the code has received more use and been further refined, it probably should be separated out into its own package as an extension of fancyvrb.

The line breaking defined here is used in minted's `minted` environment and `\mint` command, which use `Verbatim` internally. The `\mintinline` command implements line wrapping using a slightly different system (essentially, `BVerbatim`, with the `\vbox` `\let` to `\relax`). This is implemented separately within minted, rather than as an extension to fancyvrb, for simplicity and because `BVerbatim` wouldn't be itself without the box. Likewise, `breaklines` is not applied to `fancyvrb`'s `\Verb` or short verb, since their implementation is different from that of `\mintinline`. Ideally, an extension of `fancyvrb` would add line breaking to these, or (probable better) provide equivalent commands that support breaks.

9.6.1 Setup

All of the additions to fancyvrb should be defined conditionally. If an extension to fancyvrb (such as that proposed above) is loaded before minted, and if this extension provides `breaklines`, then we don't want to overwrite that definition and create a conflict. We assume that any extension of fancyvrb would use the keyval package, since that is what fancyvrb currently uses, and test for the existence of a fancyrvb keyval key `breaklines`.

```
\ifcsname KV@FV@breaklines\endcsname\else
```

9.6.2 Line breaking

Begin by defining keys, with associated macros, bools, and dimens.

FV@BreakLines Turn line breaking on of off.

```
\newboolean{FV@BreakLines}
\let\FV@ListProcessLine@Orig\FV@ListProcessLine
\define@booleankey{FV}{breaklines}%
  {\FV@BreakLinestrue
    \let\FV@ListProcessLine\FV@ListProcessLine@Break}%
  {\FV@BreakLinesfalse
    \let\FV@ListProcessLine\FV@ListProcessLine@Orig}
```

\FV@BreakIndent

```
\newdimen\FV@BreakIndent
\define@key{FV}{breakindent}{\FV@BreakIndent=#1}
\fvset{breakindent=0pt}
```

FV@BreakAutoIndent

```
\newboolean{FV@BreakAutoIndent}
\define@booleankey{FV}{breakautoindent}%
  {\FV@BreakAutoIndenttrue}{\FV@BreakAutoIndentfalse}
\fvset{breakautoindent=true}
```

\FancyVerbBreakSymbolLeft The left-hand symbol indicating a break. Since breaking is done in such a way that a left-hand symbol will often be desired while a right-hand symbol may not be, a shorthand option `breaksymbol` is supplied. This shorthand convention is continued with other options applying to the left-hand symbol.

```
\define@key{FV}{breaksymbolleft}{\def\FancyVerbBreakSymbolLeft{#1}}
\define@key{FV}{breaksymbol}{\fvset{breaksymbolleft=#1}}
\fvset{breaksymbolleft=\tiny\ensuremath{\hookrightarrow}}
```

\FancyVerbBreakSymbolRight The right-hand symbol indicating a break.

```
\define@key{FV}{breaksymbolright}{\def\FancyVerbBreakSymbolRight{#1}}
\fvset{breaksymbolright={}}
```

Separation of break symbols from the text.

\FV@BreakSymbolSepLeft

```
\newdimen\FV@BreakSymbolSepLeft
\define@key{FV}{breaksymbolsepleft}{\FV@BreakSymbolSepLeft=#1}
\define@key{FV}{breaksymbolsep}{\fvset{breaksymbolsepleft=#1}}
\fvset{breaksymbolsepleft=1em}
```

\FV@BreakSymbolSepRight

```
\newdimen\FV@BreakSymbolSepRight
\define@key{FV}{breaksymbolsepright}{\FV@BreakSymbolSepRight=#1}
\fvset{breaksymbolsepright=1em}
```

Additional indentation to make room for the break symbols.

\FV@BreakSymbolIndentLeft

```
\newdimen\FV@BreakSymbolIndentLeft
\settowidth{\FV@BreakSymbolIndentLeft}{\ttfamily xxxx}
\define@key{FV}{breaksymbolindentleft}{\FV@BreakSymbolIndentLeft=#1}
\define@key{FV}{breaksymbolindent}{\fvset{breaksymbolindentleft=#1}}
```

\FV@BreakSymbolIndentRight

```
\newdimen\FV@BreakSymbolIndentRight
\settowidth{\FV@BreakSymbolIndentRight}{\ttfamily xxxx}
\define@key{FV}{breaksymbolindentright}{\FV@BreakSymbolIndentRight=#1}
```

We need macros that contain the logic for typesetting the break symbols. By default, the symbol macros contain everything regarding the symbol and its typesetting, while these macros contain pure logic. The symbols should be wrapped in braces so that formatting commands (for example, \tiny) don't escape.

cyVerbFormatBreakSymbolLeft

```
\newcommand{\FancyVerbFormatBreakSymbolLeft}[1]{%
  \ifnum\value{linenumber}=1\relax\else{#1}\fi}
```

FancyVerbLineBreakLast We need a counter for keeping track of the internal line number for the last segment of a broken line, so that we can avoid putting a right continuation symbol there.

```
\newcounter{FancyVerbLineBreakLast}
```

\FV@SetLineBreakLast

```
\newcommand{\FV@SetLineBreakLast}{%
  \setcounter{FancyVerbLineBreakLast}{\value{linenumber}}}
```

yVerbFormatBreakSymbolRight

```
\newcommand{\FancyVerbFormatBreakSymbolRight}[1]{%
  \ifnum\value{linenumber}=\value{FancyVerbLineBreakLast}\relax\else{#1}\fi}
```

FV@BreakAnywhere Allow line breaking (almost) anywhere.

```
\newboolean{FV@BreakAnywhere}
\define@booleankey{FV}{breakanywhere}%
  {\FV@BreakAnywheretrue
    \let\FancyVerbBreakStart\FV@Break
    \let\FancyVerbBreakStop\FV@EndBreak
    \let\FV@Break@Token\FV@Break@AnyToken}%
  {\FV@BreakAnywherefalse
    \let\FancyVerbBreakStart\relax
    \let\FancyVerbBreakStop\relax}
\fvset{breakanywhere=false}
```

\FancyVerbBreakStart

```
\let\FancyVerbBreakStart\relax
```

\FancyVerbBreakStop

```
\let\FancyVerbBreakStop\relax
```

\FV@EscChars We need to define versions of common escaped characters that reduce to raw characters.

```
\edef\FV@hashchar{\string#}
\edef\FV@dollarchar{\string$}
\edef\FV@ampchar{\string&}
\edef\FV@underscorechar{\string_}
\edef\FV@tildechar{\string~}
\edef\FV@leftsquarebracket{\string[}
\edef\FV@rightsquarebracket{\string]}
\newcommand{\FV@EscChars}{%
  \let\#\FV@hashchar
  \let\%\@percentchar
  \let\{\@charlb
  \let\}\@charrb
  \let\$\FV@dollarchar
  \let\&\FV@ampchar
  \let\_\FV@underscorechar
  \let\\\@backslashchar
  \let~\FV@tildechar
  \let\~\FV@tildechar
  \let\[\FV@leftsquarebracket
  \let\]\FV@rightsquarebracket
} %$ <- highlighting
```

\FV@BreakBefore Allow line breaking (almost) anywhere, but only before specified characters.

```
\define@key{FV}{breakbefore}{%
```

```
  \ifstrempty{#1}%
   {\let\FV@BreakBefore\@empty
    \let\FancyVerbBreakStart\relax
    \let\FancyVerbBreakStop\relax}%
   {\def\FV@BreakBefore{#1}%
    \let\FancyVerbBreakStart\FV@Break
    \let\FancyVerbBreakStop\FV@EndBreak
    \let\FV@Break@Token\FV@Break@BeforeAfterToken}%
}
\fvset{breakbefore={}}
```

FV@BreakBeforeGroup Determine whether breaking before specified characters is always allowed before each individual character, or is only allowed before the first in a group of identical characters.

```
\newboolean{FV@BreakBeforeGroup}
\define@booleankey{FV}{breakbeforegroup}%
 {\FV@BreakBeforeGrouptrue}%
 {\FV@BreakBeforeGroupfalse}%
\fvset{breakbeforegroup=true}
```

\FV@BreakBeforePrep We need a way to break before characters if they have been specified as breaking characters. It would be possible to do that via a nested conditional, but that would be messy. It is much simpler to create an empty macro whose name contains the character, and test for the existence of this macro. This needs to be done inside a `\begingroup...\endgroup` so that the macros do not have to be cleaned up manually. A good place to do this is in `\FV@FormattingPrep`, which is inside a group and before processing starts. The macro is added to `\FV@FormattingPrep` below, after `\FV@BreakAfterPrep` is defined.

The procedure here is a bit roundabout. We need to use `\FV@EscChars` to handle character escapes, but the character redefinitions need to be kept local, requiring that we work within a `\begingroup...\endgroup`. So we loop through the breaking tokens and assemble a macro that will itself define character macros. Only this defining macro is declared global, and it contains *expanded* characters so that there is no longer any dependence on `\FV@EscChars`.

```
\def\FV@BreakBeforePrep{%
  \ifx\FV@BreakBefore\@empty\relax
  \else
    \gdef\FV@BreakBefore@Def{}%
    \begingroup
    \def\FV@BreakBefore@Process##1##2\FV@Undefined{%
      \expandafter\FV@BreakBefore@Process@i\expandafter{##1}%
      \expandafter\ifx\expandafter\relax\detokenize{##2}\relax
      \else
        \FV@BreakBefore@Process##2\FV@Undefined
      \fi
    }%
    \def\FV@BreakBefore@Process@i##1{%
      \g@addto@macro\FV@BreakBefore@Def{%
        \@namedef{FV@BreakBefore@Token\detokenize{##1}}{}}%
    }%
    \FV@EscChars
    \expandafter\FV@BreakBefore@Process\FV@BreakBefore\FV@Undefined
    \endgroup
    \FV@BreakBefore@Def
  \fi
}
```

\FV@BreakAfter Allow line breaking (almost) anywhere, but only after specified characters.

```
\define@key{FV}{breakafter}{%
  \ifstrempty{#1}%
   {\let\FV@BreakAfter\@empty
    \let\FancyVerbBreakStart\relax
    \let\FancyVerbBreakStop\relax}%
   {\def\FV@BreakAfter{#1}%
    \let\FancyVerbBreakStart\FV@Break
    \let\FancyVerbBreakStop\FV@EndBreak
    \let\FV@Break@Token\FV@Break@BeforeAfterToken}%
}
\fvset{breakafter={}}
```

FV@BreakAfterGroup Determine whether breaking after specified characters is always allowed after each individual character, or is only allowed after groups of identical characters.

```
\newboolean{FV@BreakAfterGroup}
\define@booleankey{FV}{breakaftergroup}%
 {\FV@BreakAfterGrouptrue}%
 {\FV@BreakAfterGroupfalse}%
\fvset{breakaftergroup=true}
```

\FV@BreakAfterPrep We need a way to break after characters if they have been specified as breaking characters. It would be possible to do that via a nested conditional, but that would be messy. It is much simpler to create an empty macro whose name contains the character, and test for the existence of this macro. This needs to be done inside a `\begingroup...\endgroup` so that the macros do not have to be cleaned up manually. A good place to do this is in `\FV@FormattingPrep`, which is inside a group and before processing starts.

The procedure here is a bit roundabout. We need to use `\FV@EscChars` to handle character escapes, but the character redefinitions need to be kept local, requiring that we work within a `\begingroup...\endgroup`. So we loop through the breaking tokens and assemble a macro that will itself define character macros. Only this defining macro is declared global, and it contains *expanded* characters so that there is no longer any dependence on `\FV@EscChars`.

```
\def\FV@BreakAfterPrep{%
  \ifx\FV@BreakAfter\@empty\relax
  \else
    \gdef\FV@BreakAfter@Def{}%
    \begingroup
    \def\FV@BreakAfter@Process##1##2\FV@Undefined{%
      \expandafter\FV@BreakAfter@Process@i\expandafter{##1}%
      \expandafter\ifx\expandafter\relax\detokenize{##2}\relax
      \else
        \FV@BreakAfter@Process##2\FV@Undefined
      \fi
    }%
    \def\FV@BreakAfter@Process@i##1{%
      \ifcsname FV@BreakBefore@Token\detokenize{##1}\endcsname
        \ifthenelse{\boolean{FV@BreakBeforeGroup}}%
         {\ifthenelse{\boolean{FV@BreakAfterGroup}}%
           {}%
           {\PackageError{minted}%
            {Conflicting breakbeforegroup and breakaftergroup for "\detokenize{##1}
            {Conflicting breakbeforegroup and breakaftergroup for "\detokenize{##1}
         {\ifthenelse{\boolean{FV@BreakAfterGroup}}%
           {\PackageError{minted}%
             {Conflicting breakbeforegroup and breakaftergroup for "\detokenize{##1
             {Conflicting breakbeforegroup and breakaftergroup for "\detokenize{##1
           {}}%
      \else
      \fi
      \g@addto@macro\FV@BreakAfter@Def{%
        \@namedef{FV@BreakAfter@Token\detokenize{##1}}{}}%
    }%
    \FV@EscChars
    \expandafter\FV@BreakAfter@Process\FV@BreakAfter\FV@Undefined
    \endgroup
    \FV@BreakAfter@Def
  \fi
}
```

Now that `\FV@BreakBeforePrep` and `\FV@BreakAfterPrep` are defined, add them to `\FV@FormattingPrep`. The ordering here is important, since `\FV@BreakAfterPrep` contains compatibility checks with `\FV@BreakBeforePrep`, and thus must be used after it.

```
\expandafter\def\expandafter\FV@FormattingPrep\expandafter{%
  \expandafter\FV@BreakBeforePrep\expandafter\FV@BreakAfterPrep\FV@FormattingPrep}
```

yVerbBreakAnywhereSymbolPre The pre-break symbol for breaks introduced by `breakanywhere`. That is, the symbol before breaks that occur between characters, rather than at spaces.

```
\define@key{FV}{breakanywheresymbolpre}{%
```

```
  \ifstrempty{#1}%
    {\def\FancyVerbBreakAnywhereSymbolPre{}}%
    {\def\FancyVerbBreakAnywhereSymbolPre{\hbox{#1}}}}
\fvset{breakanywheresymbolpre={\,\footnotesize\ensuremath{_\rfloor}}}
```

`VerbBreakAnywhereSymbolPost` The post-break symbol for breaks introduced by `breakanywhere`.

```
\define@key{FV}{breakanywheresymbolpost}{%
  \ifstrempty{#1}%
    {\def\FancyVerbBreakAnywhereSymbolPost{}}%
    {\def\FancyVerbBreakAnywhereSymbolPost{\hbox{#1}}}}
\fvset{breakanywheresymbolpost={}}
```

`ncyVerbBreakBeforeSymbolPre` The pre-break symbol for breaks introduced by `breakbefore`.

```
\define@key{FV}{breakbeforesymbolpre}{%
  \ifstrempty{#1}%
    {\def\FancyVerbBreakBeforeSymbolPre{}}%
    {\def\FancyVerbBreakBeforeSymbolPre{\hbox{#1}}}}
\fvset{breakbeforesymbolpre={\,\footnotesize\ensuremath{_\rfloor}}}
```

`cyVerbBreakBeforeSymbolPost` The post-break symbol for breaks introduced by `breakbefore`.

```
\define@key{FV}{breakbeforesymbolpost}{%
  \ifstrempty{#1}%
    {\def\FancyVerbBreakBeforeSymbolPost{}}%
    {\def\FancyVerbBreakBeforeSymbolPost{\hbox{#1}}}}
\fvset{breakbeforesymbolpost={}}
```

`ancyVerbBreakAfterSymbolPre` The pre-break symbol for breaks introduced by `breakafter`.

```
\define@key{FV}{breakaftersymbolpre}{%
  \ifstrempty{#1}%
    {\def\FancyVerbBreakAfterSymbolPre{}}%
    {\def\FancyVerbBreakAfterSymbolPre{\hbox{#1}}}}
\fvset{breakaftersymbolpre={\,\footnotesize\ensuremath{_\rfloor}}}
```

`ncyVerbBreakAfterSymbolPost` The post-break symbol for breaks introduced by `breakafter`.

```
\define@key{FV}{breakaftersymbolpost}{%
  \ifstrempty{#1}%
    {\def\FancyVerbBreakAfterSymbolPost{}}%
    {\def\FancyVerbBreakAfterSymbolPost{\hbox{#1}}}}
\fvset{breakaftersymbolpost={}}
```

`FancyVerbBreakAnywhereBreak` When `breakanywhere=true`, line breaks may occur at almost any location. This is the macro that governs the breaking in those cases. By default, `\discretionary` is used. `\discretionary` takes three arguments: a character to insert before the break, a character to insert after the break, and a character to insert if there is no break.

`\discretionary` will generally only insert breaks when breaking at spaces simply cannot make lines short enough (this may be tweaked to some extent with hyphenation settings). This can produce a somewhat ragged appearance in some cases. If you want breaks exactly at the margin (or as close as possible) regardless of whether a break at a space is an option, you may want to use `\allowbreak` instead.

```
\newcommand{\FancyVerbBreakAnywhereBreak}{%
  \discretionary{\FancyVerbBreakAnywhereSymbolPre}%
   {\FancyVerbBreakAnywhereSymbolPost}{}}
```

`\FancyVerbBreakBeforeBreak` The macro governing breaking for `breakbefore=true`.

```
\newcommand{\FancyVerbBreakBeforeBreak}{%
  \discretionary{\FancyVerbBreakBeforeSymbolPre}%
   {\FancyVerbBreakBeforeSymbolPost}{}}
```

`\FancyVerbBreakAfterBreak` The macro governing breaking for `breakafter=true`.

```
\newcommand{\FancyVerbBreakAfterBreak}{%
  \discretionary{\FancyVerbBreakAfterSymbolPre}%
   {\FancyVerbBreakAfterSymbolPost}{}}
```

Define helper macros.

`\FV@LineBox` A box for saving a line of code, so that its dimensions may be determined and thus we may figure out if it needs line breaking.

```
\newsavebox{\FV@LineBox}
```

`\FV@LineIndentBox` A box for saving the indentation of code, so that its dimensions may be determined for use in autoindentation of continuation lines.

```
\newsavebox{\FV@LineIndentBox}
```

`\FV@LineIndentChars` A macro for storing the indentation characters, if any, of a given line. For use in autoindentation of continuation lines

```
\let\FV@LineIndentChars\@empty
```

`\FV@GetLineIndent` A macro that takes a line and determines the indentation, storing the indentation chars in `\FV@LineIndentChars`.

```
\def\FV@GetNextChar{\let\FV@NextChar=}
\def\FV@CleanRemainingChars#1\FV@Undefined{}
```

```
\def\FV@GetLineIndent{\afterassignment\FV@CheckIndentChar\FV@GetNextChar}
\def\FV@CheckIndentChar{%
  \ifx\FV@NextChar\FV@Undefined
    \let\FV@Next=\relax
  \else
    \expandafter\ifx\FV@NextChar\FV@Space
      \g@addto@macro{\FV@LineIndentChars}{\FV@Space}%
      \let\FV@Next=\FV@GetLineIndent
    \else
      \expandafter\ifx\FV@NextChar\FV@Tab
        \g@addto@macro{\FV@LineIndentChars}{\FV@Tab}%
        \let\FV@Next=\FV@GetLineIndent
      \else
        \let\FV@Next=\FV@CleanRemainingChars
      \fi
    \fi
  \fi
  \FV@Next
}
```

Define the macros that actually perform `breakanywhere`, `breakbefore`, and `breakafter`.

`\FV@Break` The entry macro for breaking lines, either anywhere or before/after specified characters. The current line (or argument) will be scanned token by token/group by group, and accumulated (with added potential breaks) in `\FV@Tmp`. After scanning is complete, `\FV@Tmp` will be inserted. It would be possible to insert each token/group into the document immediately after it is scanned, instead of accumulating them in a "buffer." But that would interfere with macros. Even in the current approach, macros that take optional arguments are problematic.[7]

```
\def\FV@Break{%
  \def\FV@Tmp{}%
  \let\FV@LastToken\minted@undefined
  \FV@Break@Scan
}
```

`\FV@EndBreak`

```
\def\FV@EndBreak{\FV@Tmp}
```

`\FV@Break@Scan` Look ahead via `\@ifnextchar`. Don't do anything if we're at the end of the region to be scanned. Otherwise, invoke a macro to deal with what's next based on whether it is math, or a group, or something else.

[7]Through a suitable definition that tracks the current state and looks for square brackets, this might be circumvented. Then again, in verbatim contexts, macro use should be minimal, so the restriction to macros without optional arguments should generally not be an issue.

This and some following macros are defined inside of groups, to ensure proper catcodes.

```
\begingroup
\catcode`\$=3%
\gdef\FV@Break@Scan{%
  \@ifnextchar\FV@EndBreak%
   {}%
   {\ifx\@let@token$\relax
      \let\FV@Break@Next\FV@Break@Math
    \else
      \ifx\@let@token\bgroup\relax
        \let\FV@Break@Next\FV@Break@Group
      \else
        \let\FV@Break@Next\FV@Break@Token
      \fi
    \fi
    \FV@Break@Next}%
}
\endgroup
```

\FV@Break@Math Grab an entire math span, and insert it into `\FV@Tmp`. Due to grouping, this works even when math contains things like `\text{$x$}`. After dealing with the math span, continue scanning.

```
\begingroup
\catcode`\$=3%
\gdef\FV@Break@Math$#1${%
  \g@addto@macro{\FV@Tmp}{$#1$}%
  \let\FV@LastToken\minted@undefined
  \FV@Break@Scan}
\endgroup
```

\FV@Break@Group Grab the group, and insert it into `\FV@Tmp` (as a group) before continuing scanning.

```
\def\FV@Break@Group#1{%
  \g@addto@macro{\FV@Tmp}{{#1}}%
  \ifstrempty{#1}{}{\let\FV@LastToken\minted@undefined}%
  \FV@Break@Scan}
```

\FV@Break@Token This macro is `\let` to `\FV@Break@AnyToken` or `\FV@Break@BeforeAfterToken` by the `breakanywhere` and `breakbefore`/`breakafter` options, so it is not explicitly defined.

\FV@Break@AnyToken Deal with breaking around any token.

If it is ever necessary, it would be possible to create a more sophisticated version involving catcode checks via `\ifcat`. Something like this:

```
\begingroup
\catcode`\a=11
\catcode`\+=12
\gdef\FV@Break...
  \ifcat\noexpand#1a
    \g@addto@macro{\FV@Tmp}...
  \else
...
\endgroup
```

This doesn't break macros with *mandatory* arguments, because `\FancyVerbBreakAnywhereBreak` is inserted *before* the token. Groups themselves are added without any special handling. So a macro would end up right next to its original arguments, without anything being inserted. Optional arguments will cause this approach to fail; there is currently no attempt to identify them, since that is a much harder problem.

```
\def\FV@Break@AnyToken#1{%
  \g@addto@macro{\FV@Tmp}{\FancyVerbBreakAnywhereBreak#1}%
  \FV@Break@Scan}
```

`\FV@Break@BeforeAfterToken` Deal with breaking around only specified tokens. This is a bit trickier. We only break if a macro corresponding to the token exists. We also need to check whether the specified token should be grouped, that is, whether breaks are allowed between identical characters. All of this has to be written carefully so that nothing is accidentally inserted into the stream for future scanning.

Dealing with tokens followed by empty groups (for example, `\x{}`) is particularly challenging when we want to avoid breaks between identical characters. When a token is followed by a group, we need to save the current token for later reference (`\x` in the example), then capture and save the following group, and then—only if the group was empty—see if the following token is identical to the old saved token.

```
\def\FV@Break@BeforeAfterToken#1{%
  \ifcsname FV@BreakBefore@Token\detokenize{#1}\endcsname
    \let\FV@Break@Next\FV@Break@BeforeTokenBreak
  \else
    \ifcsname FV@BreakAfter@Token\detokenize{#1}\endcsname
      \let\FV@Break@Next\FV@Break@AfterTokenBreak
    \else
      \let\FV@Break@Next\FV@Break@BeforeAfterTokenNoBreak
    \fi
  \fi
  \FV@Break@Next{#1}%
}
\def\FV@Break@BeforeAfterTokenNoBreak#1{%
  \g@addto@macro{\FV@Tmp}{#1}%
  \let\FV@LastToken#1%
  \FV@Break@Scan}
\def\FV@Break@BeforeTokenBreak#1{%
  \ifthenelse{\boolean{FV@BreakBeforeGroup}}%
```

```
   {\ifx#1\FV@LastToken\relax
      \ifcsname FV@BreakAfter@Token\detokenize{#1}\endcsname
        \let\FV@Break@Next\FV@Break@BeforeTokenBreak@AfterRescan
        \def\FV@RescanToken{#1}%
      \else
        \g@addto@macro{\FV@Tmp}{#1}%
        \let\FV@Break@Next\FV@Break@Scan
        \let\FV@LastToken#1%
      \fi
    \else
      \ifcsname FV@BreakAfter@Token\detokenize{#1}\endcsname
        \g@addto@macro{\FV@Tmp}{\FancyVerbBreakBeforeBreak}%
        \let\FV@Break@Next\FV@Break@BeforeTokenBreak@AfterRescan
        \def\FV@RescanToken{#1}%
      \else
        \g@addto@macro{\FV@Tmp}{\FancyVerbBreakBeforeBreak#1}%
        \let\FV@Break@Next\FV@Break@Scan
        \let\FV@LastToken#1%
      \fi
    \fi}%
   {\ifcsname FV@BreakAfter@Token\detokenize{#1}\endcsname
      \g@addto@macro{\FV@Tmp}{\FancyVerbBreakBeforeBreak}%
      \let\FV@Break@Next\FV@Break@BeforeTokenBreak@AfterRescan
      \def\FV@RescanToken{#1}%
    \else
      \g@addto@macro{\FV@Tmp}{\FancyVerbBreakBeforeBreak#1}%
      \let\FV@Break@Next\FV@Break@Scan
      \let\FV@LastToken#1%
    \fi}%
  \FV@Break@Next}
\def\FV@Break@BeforeTokenBreak@AfterRescan{%
  \expandafter\FV@Break@AfterTokenBreak\FV@RescanToken}
\def\FV@Break@AfterTokenBreak#1{%
  \let\FV@LastToken#1%
  \@ifnextchar\FV@Space%
   {\g@addto@macro{\FV@Tmp}{#1}\FV@Break@Scan}%
   {\ifthenelse{\boolean{FV@BreakAfterGroup}}%
     {\ifx\@let@token#1\relax
        \g@addto@macro{\FV@Tmp}{#1}%
        \let\FV@Break@Next\FV@Break@Scan
      \else
        \ifx\@let@token\bgroup\relax
          \g@addto@macro{\FV@Tmp}{#1}%
          \let\FV@Break@Next\FV@Break@AfterTokenBreak@Group
        \else
          \g@addto@macro{\FV@Tmp}{#1\FancyVerbBreakAfterBreak}%
          \let\FV@Break@Next\FV@Break@Scan
        \fi
      \fi}%
     {\g@addto@macro{\FV@Tmp}{#1\FancyVerbBreakAfterBreak}%
```

```
      \let\FV@Break@Next\FV@Break@Scan}%
    \FV@Break@Next}%
}
\def\FV@Break@AfterTokenBreak@Group#1{%
  \g@addto@macro{\FV@Tmp}{{#1}}%
  \ifstrempty{#1}%
   {\let\FV@Break@Next\FV@Break@AfterTokenBreak@Group@i}%
   {\let\FV@Break@Next\FV@Break@Scan\let\FV@LastToken\minted@undefined}%
  \FV@Break@Next}
\def\FV@Break@AfterTokenBreak@Group@i{%
  \@ifnextchar\FV@LastToken%
   {\FV@Break@Scan}%
   {\g@addto@macro{\FV@Tmp}{\FancyVerbBreakAfterBreak}%
    \FV@Break@Scan}}
```

And finally the really important things.

`\FV@makeLineNumber` We need a version of lineno's `\makeLineNumber` that is adapted for our purposes. This is adapted directly from the example `\makeLineNumber` that is given in the lineno documentation under the discussion of internal line numbers. The `\FV@SetLineBreakLast` is needed to determine the internal line number of the last segment of the broken line, so that we can disable the right-hand break symbol on this segment. When a right-hand break symbol is in use, a line of code will be processed twice: once to determine the last internal line number, and once to use this information only to insert right-hand break symbols on the appropriate lines. During the second run, `\FV@SetLineBreakLast` is disabled by `\let`ting it to `\relax`.

```
\def\FV@makeLineNumber{%
  \hss
  \FancyVerbFormatBreakSymbolLeft{\FancyVerbBreakSymbolLeft}%
  \hbox to \FV@BreakSymbolSepLeft{\hfill}%
  \rlap{\hskip\linewidth
    \hbox to \FV@BreakSymbolSepRight{\hfill}%
    \FancyVerbFormatBreakSymbolRight{\FancyVerbBreakSymbolRight}%
    \FV@SetLineBreakLast
  }%
}
```

`\FV@SaveLineBox` This is the macro that does most of the work. This was inspired by Marco Daniel's code at http://tex.stackexchange.com/a/112573/10742.

This macro is invoked when a line is too long. We modify the `\linewidth` to take into account `breakindent` and `breakautoindent`, and insert `\hbox`es to fill the empty space. We also account for `breaksymbolindentleft` and `breaksymbolindentright`, but *only* when there are actually break symbols. The code is placed in a `\parbox`. Break symbols are inserted via lineno's `internallinenumbers*`, which does internal line numbers without continuity between environments (the `linenumber` counter is automatically reset). The beginning of the code has negative `\hspace` inserted to pull it out to the correct starting position. `\strut`s are used to maintain correct line heights. The `\parbox` is followed by an empty `\hbox` that takes up the space needed for a right-hand break symbol (if any).

```
\def\FV@SaveLineBox#1{%
  \savebox{\FV@LineBox}{%
    \advance\linewidth by -\FV@BreakIndent
    \hbox to \FV@BreakIndent{\hfill}%
    \ifthenelse{\boolean{FV@BreakAutoIndent}}%
     {\let\FV@LineIndentChars\@empty
      \FV@GetLineIndent#1\FV@Undefined
      \savebox{\FV@LineIndentBox}{\FV@LineIndentChars}%
      \hbox to \wd\FV@LineIndentBox{\hfill}%
      \advance\linewidth by -\wd\FV@LineIndentBox}%
     {}%
    \ifdefempty{\FancyVerbBreakSymbolLeft}{}%
     {\hbox to \FV@BreakSymbolIndentLeft{\hfill}%
      \advance\linewidth by -\FV@BreakSymbolIndentLeft}%
    \ifdefempty{\FancyVerbBreakSymbolRight}{}%
     {\advance\linewidth by -\FV@BreakSymbolIndentRight}%
    \parbox[t]{\linewidth}{%
      \raggedright
      \leftlinenumbers*
      \begin{internallinenumbers*}%
      \let\makeLineNumber\FV@makeLineNumber
      \noindent\hspace*{-\FV@BreakIndent}%
      \ifdefempty{\FancyVerbBreakSymbolLeft}{}{%
        \hspace*{-\FV@BreakSymbolIndentLeft}}%
      \ifthenelse{\boolean{FV@BreakAutoIndent}}%
       {\hspace*{-\wd\FV@LineIndentBox}}%
       {}%
      \strut\FancyVerbFormatText{%
        \FancyVerbBreakStart#1\FancyVerbBreakStop}\nobreak\strut
      \end{internallinenumbers*}
    }%
    \ifdefempty{\FancyVerbBreakSymbolRight}{}%
     {\hbox to \FV@BreakSymbolIndentRight{\hfill}}%
  }%
}
```

`\FancyVerbFormatText` The introduction of line breaks introduces an issue for `\FancyVerbFormatLine`. Does it format the entire line (outside the `\parbox`), or only the text part of the line (inside the `\parbox`)? Since both might be desirable, `\FancyVerbFormatLine` is assigned to the entire line, and a new macro `\FancyVerbFormatText` is assigned to the text, within the `\parbox`.

```
\def\FancyVerbFormatText#1{#1}
```

\FV@ListProcessLine@Break This macro is based on \FV@ListProcessLine and follows it as closely as possible. The \linewidth is reduced by \FV@FrameSep and \FV@FrameRule so that text will not overrun frames. This is done conditionally based on which frames are in use. We save the current line in a box, and only do special things if the box is too wide. For uniformity, all text is placed in a \parbox, even if it doesn't need to be wrapped.

If a line is too wide, then it is passed to \FV@SaveLineBox. If there is no right-hand break symbol, then the saved result in \FV@LineBox may be used immediately. If there is a right-hand break symbol, then the line must be processed a second time, so that the right-hand break symbol may be removed from the final segment of the broken line (since it does not continue). During the first use of \FV@SaveLineBox, the counter FancyVerbLineBreakLast is set to the internal line number of the last segment of the broken line. During the second use of \FV@SaveLineBox, we disable this (\let\FV@SetLineBreakLast\relax) so that the value of FancyVerbLineBreakLast remains fixed and thus may be used to determine when a right-hand break symbol should be inserted.

```
\def\FV@ListProcessLine@Break#1{%
  \ifx\FV@ObeyTabsInit\relax\else
    \PackageError{minted}%
     {the options obeytabs and breaklines are not compatible}{}%
  \fi
  \hbox to \hsize{%
  \kern\leftmargin
  \hbox to \linewidth{%
  \ifx\FV@RightListFrame\relax\else
    \advance\linewidth by -\FV@FrameSep
    \advance\linewidth by -\FV@FrameRule
  \fi
  \ifx\FV@LeftListFrame\relax\else
    \advance\linewidth by -\FV@FrameSep
    \advance\linewidth by -\FV@FrameRule
  \fi
  \sbox{\FV@LineBox}{\FancyVerbFormatLine{\FancyVerbFormatText{#1}}}%
  \ifdim\wd\FV@LineBox>\linewidth
    \setcounter{FancyVerbLineBreakLast}{0}%
    \FV@SaveLineBox{#1}%
    \ifdefempty{\FancyVerbBreakSymbolRight}{}{%
      \let\FV@SetLineBreakLast\relax
      \FV@SaveLineBox{#1}}%
    \FV@LeftListNumber
    \FV@LeftListFrame
    \FancyVerbFormatLine{\usebox{\FV@LineBox}}%
    \FV@RightListFrame
    \FV@RightListNumber
  \else
    \FV@LeftListNumber
    \FV@LeftListFrame
    \FancyVerbFormatLine{%
      \parbox[t]{\linewidth}{\noindent\strut\FancyVerbFormatText{#1}\strut}}%
    \FV@RightListFrame
    \FV@RightListNumber
  \fi}%
  \hss}\baselineskip\z@\lineskip\z@}
```

9.7 `linenos`

Since fancyvrb currently doesn't have a `linenos` key, we create one that mimics `numbers=left` (but only after checking to make sure that another package hasn't already patched this).

```
\ifcsname KV@FV@linenos\endcsname\else
\define@booleankey{FV}{linenos}%
  {\@nameuse{FV@Numbers@left}}{\@nameuse{FV@Numbers@none}}
\fi
```

9.8 Cleanup

Finally, end the conditional creation of fancyvrb extensions.

```
\fi
```

9.9 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 tcolorbox or mdframed.

First, we need to define a new save box.

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

Now we can define the environment that applies a background color. Prior to minted 2.2, this involved a `minipage`. However, that approach was problematic because it did not allow linebreaks, would be pushed into the margin by immediately preceding text, and had very different whitespace separation from preceding and following text compared to no background color. In version 2.2, this was replaced with an approach based on framed. `\FV@NumberSep` is adjusted by `\fboxsep` to ensure that line numbers remain in the same location in the margin regardless of whether `bgcolor` is used.

```
\newenvironment{minted@colorbg}[1]{%
```

```
  \setlength{\OuterFrameSep}{0pt}%
  \colorlet{shadecolor}{#1}%
  \let\minted@tmp\FV@NumberSep
  \edef\FV@NumberSep{%
    \the\numexpr\dimexpr\minted@tmp+\number\fboxsep\relax sp\relax}%
  \medskip
  \begin{snugshade*}}
 {\end{snugshade*}%
  \medskip\noindent}
```

`\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\minted@jobname.pyg\relax
  \immediate\write\minted@code{\expandafter\detokenize\expandafter{#1}}%
  \immediate\closeout\minted@code}
```

`minted@FancyVerbLineTemp` At various points, we will need a temporary counter for storing and then restoring the value of `FancyVerbLine`. When using the `langlinenos` option, we need to store the current value of `FancyVerbLine`, then set `FancyVerbLine` to the current value of a language-specific counter, and finally restore `FancyVerbLine` to its initial value after the current chunk of code has been typeset. In patching `VerbatimOut`, we need to prevent `FancyVerbLine` from being incremented during the write process.

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

`\minted@FVB@VerbatimOut` We need a custom version of fancyvrb's `\FVB@VerbatimOut` that supports Unicode (everything written to file is `\detokenized`). We also need to prevent the value of `FancyVerbLine` from being incorrectly incremented.

```
\newcommand{\minted@write@detok}[1]{%
  \immediate\write\FV@OutFile{\detokenize{#1}}}
\newcommand{\minted@FVB@VerbatimOut}[1]{%
  \setcounter{minted@FancyVerbLineTemp}{\value{FancyVerbLine}}%
  \@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@FVE@VerbatimOut Likewise, we need a custom version of `\FVE@VerbatimOut` that completes the protection of `FancyVerbLine` from being incremented.

```
\newcommand{\minted@FVE@VerbatimOut}{%
  \immediate\closeout\FV@OutFile\endgroup\@esphack
  \setcounter{FancyVerbLine}{\value{minted@FancyVerbLineTemp}}}%
```

\MintedPygmentize We need a way to customize the executable/script that is called to perform highlighting. Typically, we will want `pygmentize`. But advanced users might wish to use a custom Python script instead. The command is only defined if it does not exist. In general, the command should be `\renewcommand`ed after the package is loaded, but this way, it will work if defined before minted is loaded.

```
\ifcsname MintedPygmentize\endcsname\else
  \newcommand{\MintedPygmentize}{pygmentize}
\fi
```

minted@pygmentizecounter We need a counter to keep track of how many files have been pygmentized. This is primarily used with `finalizecache` for naming cache files sequentially in `listing<number>.pygtex` form.

```
\newcounter{minted@pygmentizecounter}
```

\minted@pygmentize Pygmentize a file (default: `\minted@outputdir\minted@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.[8] 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

[8]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.

is actually saved by Pygments is identical. The difference in final appearance is due to how the environment and command treat the Pygments output.

This macro must always be checked carefully whenever it is modified. Under no circumstances should `#1` be written to or opened by Python in write mode. When `\inputminted` is used, `#1` will be an external file that is brought in for highlighting, so it must be left intact.

```
\newcommand{\minted@pygmentize}[2][\minted@outputdir\minted@jobname.pyg]{%
  \stepcounter{minted@pygmentizecounter}%
  \ifthenelse{\equal{\minted@get@opt{autogobble}{false}}{true}}%
    {\def\minted@codefile{\minted@outputdir\minted@jobname.pyg}}%
    {\def\minted@codefile{#1}}%
  \ifthenelse{\boolean{minted@isinline}}%
    {\def\minted@optlistcl@inlines{%
      \minted@optlistcl@g@i
      \csname minted@optlistcl@lang\minted@lang @i\endcsname}}%
    {\let\minted@optlistcl@inlines\@empty}%
  \def\minted@cmd{%
    \ifminted@kpsewhich\ifwindows powershell\space\fi\fi
    \MintedPygmentize\space -l #2
    -f latex -P commandprefix=PYG -F tokenmerge
    \minted@optlistcl@g \csname minted@optlistcl@lang\minted@lang\endcsname
    \minted@optlistcl@inlines
    \minted@optlistcl@cmd -o \minted@outputdir\minted@infile\space
    \ifminted@kpsewhich
      \ifwindows
        \detokenize{$}(kpsewhich \minted@codefile)%
      \else
        \detokenize{`}kpsewhich \minted@codefile\space
          \detokenize{||} \minted@codefile\detokenize{`}%
      \fi
    \else
      \minted@codefile
    \fi}%
  % For debugging, uncomment: %%%%
  % \immediate\typeout{\minted@cmd}%
  % %%%%
  \ifthenelse{\boolean{minted@cache}}%
    {%
      \ifminted@frozencache
      \else
        \ifx\XeTeXinterchartoks\minted@undefined
          \ifthenelse{\equal{\minted@get@opt{autogobble}{false}}{true}}%
            {\edef\minted@hash{\pdf@filemdfivesum{#1}%
              \pdf@mdfivesum{\minted@cmd autogobble}}}%
            {\edef\minted@hash{\pdf@filemdfivesum{#1}%
              \pdf@mdfivesum{\minted@cmd}}}%
        \else
          \immediate\openout\minted@code\minted@jobname.mintedcmd\relax
          \immediate\write\minted@code{\minted@cmd}%
          \ifthenelse{\equal{\minted@get@opt{autogobble}{false}}{true}}%
            {\immediate\write\minted@code{autogobble}}{}%
          \immediate\closeout\minted@code
          \edef\minted@argone@esc{#1}%
          \StrSubstitute{\minted@argone@esc}{\@backslashchar}{\@backslashchar\@back
          \StrSubstitute{\minted@argone@esc}{"}{\@backslashchar"}[\minted@argone@esc
          \edef\minted@tmpfname@esc{\minted@outputdir\minted@jobname}%
          \StrSubstitute{\minted@tmpfname@esc}{\@backslashchar}{\@backslashchar\@bac
          \StrSubstitute{\minted@tmpfname@esc}{"}{\@backslashchar"}[\minted@tmpfname
          %Cheating a little here by using ASCII codes to write `{` and `}`
          %in the Python code
          \def\minted@hashcmd{%
            \detokenize{python -c "import hashlib; import os;
              hasher = hashlib.sha1();
              f = open(os.path.expanduser(os.path.expandvars(\"}\minted@tmpfname@esc
              hasher.update(f.read());
              f.close();
              f = open(os.path.expanduser(os.path.expandvars(\"}\minted@argone@esc\
              hasher.update(f.read());
              f.close();
              f = open(os.path.expanduser(os.path.expandvars(\"}\minted@tmpfname@esc
              macro = \"\\edef\\minted@hash\" + chr(123) + hasher.hexdigest() + chr
              f.write(\"\\makeatletter\" + macro + \"\\makeatother\\endinput\n\");
              f.close();"}}%
          \ShellEscape{\minted@hashcmd}%
          \minted@input{\minted@outputdir\minted@jobname.mintedmd5}%
        \fi
        \edef\minted@infile{\minted@cachedir/\minted@hash.pygtex}%
        \IfFileExists{\minted@infile}{}{%
          \ifthenelse{\equal{\minted@get@opt{autogobble}{false}}{true}}{%
            \edef\minted@argone@esc{#1}%
            \StrSubstitute{\minted@argone@esc}{\@backslashchar}{\@backslashchar\@bac
            \StrSubstitute{\minted@argone@esc}{"}{\@backslashchar"}[\minted@argone@
            \edef\minted@tmpfname@esc{\minted@outputdir\minted@jobname}%
            \StrSubstitute{\minted@tmpfname@esc}{\@backslashchar}{\@backslashchar\@
            \StrSubstitute{\minted@tmpfname@esc}{"}{\@backslashchar"}[\minted@tmpfn
            %Need a version of open() that supports encoding under Python 2
            \edef\minted@autogobblecmd{%
              \detokenize{python -c "import sys; import os;
              import textwrap;
              from io import open;
              f = open(os.path.expanduser(os.path.expandvars(\"}\minted@argone@esc\
              t = f.read();
              f.close();
              f = open(os.path.expanduser(os.path.expandvars(\"}\minted@tmpfname@esc
              f.write(textwrap.dedent(t));
              f.close();"}%
            }%
            \ShellEscape{\minted@autogobblecmd}}{}%
          \ShellEscape{\minted@cmd}}%
      \fi
      \ifthenelse{\boolean{minted@finalizecache}}%
       {%
          \edef\minted@cachefilename{listing\arabic{minted@pygmentizecounter}.pygte
          \edef\minted@actualinfile{\minted@cachedir/\minted@cachefilename}%
          \ifwindows
            \StrSubstitute{\minted@infile}{/}{\@backslashchar}[\minted@infile@windo
            \StrSubstitute{\minted@actualinfile}{/}{\@backslashchar}[\minted@actual
            \ShellEscape{move /y \minted@infile@windows\space\minted@actualinfile@w
          \else
            \ShellEscape{mv -f \minted@infile\space\minted@actualinfile}%
          \fi
          \let\minted@infile\minted@actualinfile
          \expandafter\minted@addcachefile\expandafter{\minted@cachefilename}%
       }%
       {\ifthenelse{\boolean{minted@frozencache}}%
         {%
            \edef\minted@cachefilename{listing\arabic{minted@pygmentizecounter}.pyg
            \edef\minted@infile{\minted@cachedir/\minted@cachefilename}%
            \expandafter\minted@addcachefile\expandafter{\minted@cachefilename}}%
         {\expandafter\minted@addcachefile\expandafter{\minted@hash.pygtex}}%
       }%
      \minted@inputpyg}%
    {%
      \ifthenelse{\equal{\minted@get@opt{autogobble}{false}}{true}}{%
        \edef\minted@argone@esc{#1}%
        \StrSubstitute{\minted@argone@esc}{\@backslashchar}{\@backslashchar\@backsl
        \StrSubstitute{\minted@argone@esc}{"}{\@backslashchar"}[\minted@argone@esc]
        \edef\minted@tmpfname@esc{\minted@outputdir\minted@jobname}%
        \StrSubstitute{\minted@tmpfname@esc}{\@backslashchar}{\@backslashchar\@back
        \StrSubstitute{\minted@tmpfname@esc}{"}{\@backslashchar"}[\minted@tmpfname@
        %Need a version of open() that supports encoding under Python 2
        \edef\minted@autogobblecmd{%
          \detokenize{python -c "import sys; import os;
          import textwrap;
          from io import open;
          f = open(os.path.expanduser(os.path.expandvars(\"}\minted@argone@esc\deto
          t = f.read();
          f.close();
          f = open(os.path.expanduser(os.path.expandvars(\"}\minted@tmpfname@esc.py
          f.write(textwrap.dedent(t));
          f.close();"}%
        }%
        \ShellEscape{\minted@autogobblecmd}}{}%
      \ShellEscape{\minted@cmd}%
      \minted@inputpyg}%
}
```

\minted@inputpyg For increased clarity, the actual \input process is separated out into its own macro. The bgcolor option needs to be dealt with in different ways depending on whether we are using \mintinline. It is simplest to apply this option here, so that the macro redefinitions may be local and thus do not need to be manually reset later. \FV@Space is also patched for math mode, so that space characters will vanish rather than appear as literal spaces within math mode. To simplify the logic, breakbytoken is turned on if breakbytokenanywhere is on.

At the last possible moment, \PYG is \let to \PYG<style>. All modifications to the style macro for breaking are made to \PYG<style> rather than \PYG, so that the \leting that must ultimately take place will indeed do what is intended.

```
\def\FV@SpaceMMode{ }
\def\minted@BreakBeforePrep@extension{%
  \ifcsname FV@BreakBefore@Token\@backslashchar\endcsname
    \@namedef{FV@BreakBefore@Token\detokenize{\PYGZbs}}{}%
  \fi
  \ifcsname FV@BreakBefore@Token\FV@underscorechar\endcsname
    \@namedef{FV@BreakBefore@Token\detokenize{\PYGZus}}{}%
  \fi
  \ifcsname FV@BreakBefore@Token\@charlb\endcsname
    \@namedef{FV@BreakBefore@Token\detokenize{\PYGZob}}{}%
  \fi
  \ifcsname FV@BreakBefore@Token\@charrb\endcsname
    \@namedef{FV@BreakBefore@Token\detokenize{\PYGZcb}}{}%
  \fi
  \ifcsname FV@BreakBefore@Token\detokenize{^}\endcsname
    \@namedef{FV@BreakBefore@Token\detokenize{\PYGZca}}{}%
  \fi
  \ifcsname FV@BreakBefore@Token\FV@ampchar\endcsname
    \@namedef{FV@BreakBefore@Token\detokenize{\PYGZam}}{}%
  \fi
  \ifcsname FV@BreakBefore@Token\detokenize{<}\endcsname
    \@namedef{FV@BreakBefore@Token\detokenize{\PYGZlt}}{}%
  \fi
  \ifcsname FV@BreakBefore@Token\detokenize{>}\endcsname
    \@namedef{FV@BreakBefore@Token\detokenize{\PYGZgt}}{}%
  \fi
  \ifcsname FV@BreakBefore@Token\FV@hashchar\endcsname
    \@namedef{FV@BreakBefore@Token\detokenize{\PYGZsh}}{}%
  \fi
  \ifcsname FV@BreakBefore@Token\@percentchar\endcsname
    \@namedef{FV@BreakBefore@Token\detokenize{\PYGZpc}}{}%
  \fi
  \ifcsname FV@BreakBefore@Token\FV@dollarchar\endcsname
    \@namedef{FV@BreakBefore@Token\detokenize{\PYGZdl}}{}%
  \fi
  \ifcsname FV@BreakBefore@Token\detokenize{-}\endcsname
    \@namedef{FV@BreakBefore@Token\detokenize{\PYGZhy}}{}%
  \fi
  \ifcsname FV@BreakBefore@Token\detokenize{'}\endcsname
    \@namedef{FV@BreakBefore@Token\detokenize{\PYGZsq}}{}%
  \fi
  \ifcsname FV@BreakBefore@Token\detokenize{"}\endcsname
    \@namedef{FV@BreakBefore@Token\detokenize{\PYGZdq}}{}%
  \fi
  \ifcsname FV@BreakBefore@Token\FV@tildechar\endcsname
    \@namedef{FV@BreakBefore@Token\detokenize{\PYGZti}}{}%
  \fi
  \ifcsname FV@BreakBefore@Token\detokenize{@}\endcsname
    \@namedef{FV@BreakBefore@Token\detokenize{\PYGZat}}{}%
  \fi
  \ifcsname FV@BreakBefore@Token\detokenize{[}\endcsname
    \@namedef{FV@BreakBefore@Token\detokenize{\PYGZlb}}{}%
  \fi
  \ifcsname FV@BreakBefore@Token\detokenize{]}\endcsname
    \@namedef{FV@BreakBefore@Token\detokenize{\PYGZrb}}{}%
  \fi
}
\def\minted@BreakAfterPrep@extension{%
  \ifcsname FV@BreakAfter@Token\@backslashchar\endcsname
    \@namedef{FV@BreakAfter@Token\detokenize{\PYGZbs}}{}%
  \fi
  \ifcsname FV@BreakAfter@Token\FV@underscorechar\endcsname
    \@namedef{FV@BreakAfter@Token\detokenize{\PYGZus}}{}%
  \fi
  \ifcsname FV@BreakAfter@Token\@charlb\endcsname
    \@namedef{FV@BreakAfter@Token\detokenize{\PYGZob}}{}%
  \fi
  \ifcsname FV@BreakAfter@Token\@charrb\endcsname
    \@namedef{FV@BreakAfter@Token\detokenize{\PYGZcb}}{}%
  \fi
  \ifcsname FV@BreakAfter@Token\detokenize{^}\endcsname
    \@namedef{FV@BreakAfter@Token\detokenize{\PYGZca}}{}%
  \fi
  \ifcsname FV@BreakAfter@Token\FV@ampchar\endcsname
    \@namedef{FV@BreakAfter@Token\detokenize{\PYGZam}}{}%
  \fi
  \ifcsname FV@BreakAfter@Token\detokenize{<}\endcsname
    \@namedef{FV@BreakAfter@Token\detokenize{\PYGZlt}}{}%
  \fi
  \ifcsname FV@BreakAfter@Token\detokenize{>}\endcsname
    \@namedef{FV@BreakAfter@Token\detokenize{\PYGZgt}}{}%
  \fi
  \ifcsname FV@BreakAfter@Token\FV@hashchar\endcsname
    \@namedef{FV@BreakAfter@Token\detokenize{\PYGZsh}}{}%
  \fi
  \ifcsname FV@BreakAfter@Token\@percentchar\endcsname
    \@namedef{FV@BreakAfter@Token\detokenize{\PYGZpc}}{}%
  \fi
  \ifcsname FV@BreakAfter@Token\FV@dollarchar\endcsname
    \@namedef{FV@BreakAfter@Token\detokenize{\PYGZdl}}{}%
  \fi
  \ifcsname FV@BreakAfter@Token\detokenize{-}\endcsname
    \@namedef{FV@BreakAfter@Token\detokenize{\PYGZhy}}{}%
  \fi
  \ifcsname FV@BreakAfter@Token\detokenize{'}\endcsname
    \@namedef{FV@BreakAfter@Token\detokenize{\PYGZsq}}{}%
  \fi
  \ifcsname FV@BreakAfter@Token\detokenize{"}\endcsname
    \@namedef{FV@BreakAfter@Token\detokenize{\PYGZdq}}{}%
  \fi
  \ifcsname FV@BreakAfter@Token\FV@tildechar\endcsname
    \@namedef{FV@BreakAfter@Token\detokenize{\PYGZti}}{}%
  \fi
  \ifcsname FV@BreakAfter@Token\detokenize{@}\endcsname
    \@namedef{FV@BreakAfter@Token\detokenize{\PYGZat}}{}%
  \fi
  \ifcsname FV@BreakAfter@Token\detokenize{[}\endcsname
    \@namedef{FV@BreakAfter@Token\detokenize{\PYGZlb}}{}%
  \fi
  \ifcsname FV@BreakAfter@Token\detokenize{]}\endcsname
    \@namedef{FV@BreakAfter@Token\detokenize{\PYGZrb}}{}%
  \fi
}
\newcommand{\minted@inputpyg}{%
  \let\FV@BreakBeforePrep@orig\FV@BreakBeforePrep
  \def\FV@BreakBeforePrep{%
    \FV@BreakBeforePrep@orig\minted@BreakBeforePrep@extension}%
  \let\FV@BreakAfterPrep@orig\FV@BreakAfterPrep
  \def\FV@BreakAfterPrep{%
    \FV@BreakAfterPrep@orig\minted@BreakAfterPrep@extension}%
  \everymath\expandafter{\the\everymath\let\FV@Space\FV@SpaceMMode}%
  \ifthenelse{\equal{\minted@get@opt{breakbytokenanywhere}{false}}{true}}%
    {\setkeys{minted@opt@cmd}{breakbytoken=true}}{}%
  \ifthenelse{\boolean{FV@BreakAnywhere}}%
    {\expandafter\let\expandafter\minted@orig@PYG@breakanywhere%
       \csname PYG\minted@get@opt{style}{default}\endcsname
     \expandafter\def\csname PYG\minted@get@opt{style}{default}\endcsname##1##2{%
       \minted@orig@PYG@breakanywhere{##1}%
         {\FancyVerbBreakStart##2\FancyVerbBreakStop}}}{}%
  \ifx\FV@BreakBefore\@empty
    \ifx\FV@BreakAfter\@empty
    \else
      \expandafter\let\expandafter\minted@orig@PYG@breakbeforeafter%
        \csname PYG\minted@get@opt{style}{default}\endcsname
      \expandafter\def\csname PYG\minted@get@opt{style}{default}\endcsname##1##2{%
        \minted@orig@PYG@breakbeforeafter{##1}%
         {\FancyVerbBreakStart##2\FancyVerbBreakStop}}%
    \fi
  \else
    \expandafter\let\expandafter\minted@orig@PYG@breakbeforeafter%
      \csname PYG\minted@get@opt{style}{default}\endcsname
    \expandafter\def\csname PYG\minted@get@opt{style}{default}\endcsname##1##2{%
      \minted@orig@PYG@breakbeforeafter{##1}%
       {\FancyVerbBreakStart##2\FancyVerbBreakStop}}%
  \fi
  \ifthenelse{\boolean{minted@isinline}}%
   {\ifthenelse{\equal{\minted@get@opt{breaklines}{false}}{true}}%
    {\let\FV@BeginVBox\relax
     \let\FV@EndVBox\relax
     \def\FV@BProcessLine##1{\FancyVerbFormatLine{##1}}%
     \ifthenelse{\equal{\minted@get@opt{breakbytoken}{false}}{true}}%
      {\minted@inputpyg@breakbytoken
       \minted@inputpyg@inline}%
      {\minted@inputpyg@inline}}%
    {\minted@inputpyg@inline}}%
   {\ifthenelse{\equal{\minted@get@opt{breaklines}{false}}{true}}%
     {\ifthenelse{\equal{\minted@get@opt{breakbytoken}{false}}{true}}%
       {\minted@inputpyg@breakbytoken
        \minted@inputpyg@block}%
       {\minted@inputpyg@block}}%
     {\minted@inputpyg@block}}%
}
\def\minted@inputpyg@breakbytoken{%
  \expandafter\let\expandafter\minted@orig@PYG@breakbytoken%
    \csname PYG\minted@get@opt{style}{default}\endcsname
  \ifthenelse{\equal{\minted@get@opt{breakbytokenanywhere}{false}}{true}}%
   {\let\minted@orig@allowbreak\allowbreak
    \def\allowbreak{\let\allowbreak\minted@orig@allowbreak}%
    \expandafter\def\csname PYG\minted@get@opt{style}{default}\endcsname##1##2{%
      \allowbreak{}\leavevmode\hbox{\minted@orig@PYG@breakbytoken{##1}{##2}}}}%
   {\expandafter\def\csname PYG\minted@get@opt{style}{default}\endcsname##1##2{%
      \leavevmode\hbox{\minted@orig@PYG@breakbytoken{##1}{##2}}}}%
}
\def\minted@inputpyg@inline{%
  \expandafter\let\expandafter\PYG%
    \csname PYG\minted@get@opt{style}{default}\endcsname
  \ifthenelse{\equal{\minted@get@opt{bgcolor}{}}{}}%
   {\minted@input{\minted@outputdir\minted@infile}}%
   {\colorbox{\minted@get@opt{bgcolor}{}}{%
      \minted@input{\minted@outputdir\minted@infile}}}%
}
\def\minted@inputpyg@block{%
  \expandafter\let\expandafter\PYG%
    \csname PYG\minted@get@opt{style}{default}\endcsname
  \ifthenelse{\equal{\minted@get@opt{bgcolor}{}}{}}%
   {\minted@input{\minted@outputdir\minted@infile}}%
   {\begin{minted@colorbg}{\minted@get@opt{bgcolor}{}}%
    \minted@input{\minted@outputdir\minted@infile}%
    \end{minted@colorbg}}}
```

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

`\minted@langlinenoson`

```
\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
}
```

9.10 Public API

`\setminted` Set global or language-level options.

```
\newcommand{\setminted}[2][]{%
  \ifthenelse{\equal{#1}{}}%
    {\setkeys{minted@opt@g}{#2}}%
    {\minted@configlang{#1}%
      \setkeys{minted@opt@lang}{#2}}}
```

`\setmintedinline` Set global or language-level options, but only for inline (`\mintinline`) content. These settings will override the corresponding `\setminted` settings.

```
\newcommand{\setmintedinline}[2][]{%
  \ifthenelse{\equal{#1}{}}%
    {\setkeys{minted@opt@g@i}{#2}}%
    {\minted@configlang{#1}%
      \setkeys{minted@opt@lang@i}{#2}}}
```

Now that the settings macros exist, we go ahead and create any needed defaults.

```
\setmintedinline[php]{startinline=true}
```

\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}}
```

\minted@defwhitespace@retok The `\mint` and `\mintinline` commands need to be able to retokenize the code they collect, particularly in `draft` mode. Retokenizeation involves expansion combined with `\scantokens`, with active space and tab characters. The active characters need to expand to the appropriate fancyvrb macros, but the macros themselves should not be expanded. We need a macro that will accomplish the appropriate definitions.

```
\begingroup
\catcode`\ =\active
\catcode`\^^I=\active
\gdef\minted@defwhitespace@retok{\def {\noexpand\FV@Space}\def^^I{\noexpand\FV@Tab}
\endgroup
```

\minted@writecmdcode The `\mintinline` and `\mint` commands will need to write the code they capture to a temporary file for highlighting. It will be convenient to be able to accomplish this via a simple macro, since that makes it simpler to deal with any expansion of what is to be written. This isn't needed for the `minted` environment, because the (patched) `VerbatimOut` is used.

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

\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.

Everything needs to be within a `\begingroup...\endgroup` to prevent settings from escaping.

In the case of `draft` mode, the code is captured and retokenized. Then the internals of fancyvrb are used to emulate `SaveVerbatim`, so that `\BUseVerbatim` may be employed.

The FancyVerbLine counter is altered somehow within \minted@pygmentize, so we protect against this.

```
\newrobustcmd{\mintinline}[2][]{%
  \begingroup
  \setboolean{minted@isinline}{true}%
  \minted@configlang{#2}%
  \setkeys{minted@opt@cmd}{#1}%
  \minted@fvset
  \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
  \catcode`\^^I=\active
  \minted@inline@ii}
\ifthenelse{\boolean{minted@draft}}%
  {\newcommand{\minted@inline@iii}[1]{%
    \endgroup
    \begingroup
    \minted@defwhitespace@retok
    \everyeof{\noexpand}%
    \endlinechar-1\relax
    \let\do\@makeother\dospecials
    \catcode`\ =\active
    \catcode`\^^I=\active
    \xdef\minted@tmp{\scantokens{#1}}%
    \endgroup
    \let\FV@Line\minted@tmp
    \def\FV@SV@minted@tmp{%
      \FV@Gobble
      \expandafter\FV@ProcessLine\expandafter{\FV@Line}}%
    \ifthenelse{\equal{\minted@get@opt{breaklines}{false}}{true}}%
     {\let\FV@BeginVBox\relax
      \let\FV@EndVBox\relax
      \def\FV@BProcessLine##1{\FancyVerbFormatLine{##1}}%
      \BUseVerbatim{minted@tmp}}%
     {\BUseVerbatim{minted@tmp}}%
    \endgroup}}%
  {\newcommand{\minted@inline@iii}[1]{%
```

```
    \endgroup
    \minted@writecmdcode{#1}%
    \RecustomVerbatimEnvironment{Verbatim}{BVerbatim}{}%
    \setcounter{minted@FancyVerbLineTemp}{\value{FancyVerbLine}}%
    \minted@pygmentize{\minted@lang}%
    \setcounter{FancyVerbLine}{\value{minted@FancyVerbLineTemp}}%
    \endgroup}}
```

`\mint` Highlight a small piece of verbatim code (a single line).

The `draft` version digs into a good deal of fancyvrb internals. We want to employ `\UseVerbatim`, and this requires assembling a macro equivalent to what `SaveVerbatim` would have created. Actually, this is superior to what `SaveVerbatim` would yield, because line numbering is handled correctly.

```
\newrobustcmd{\mint}[2][]{%
  \begingroup
  \minted@configlang{#2}%
  \setkeys{minted@opt@cmd}{#1}%
  \minted@fvset
  \begingroup
  \let\do\@makeother\dospecials
  \catcode`\{=1
  \catcode`\}=2
  \catcode`\^^I=\active
  \@ifnextchar\bgroup
    {\mint@iii}%
    {\catcode`\{=12\catcode`\}=12
      \mint@i}}
\def\mint@i#1{%
  \endgroup
  \def\mint@ii##1#1{%
    \mint@iii{##1}}%
  \begingroup
  \let\do\@makeother\dospecials
  \catcode`\^^I=\active
  \mint@ii}
\ifthenelse{\boolean{minted@draft}}%
  {\newcommand{\mint@iii}[1]{%
    \endgroup
    \begingroup
    \minted@defwhitespace@retok
    \everyeof{\noexpand}%
    \endlinechar-1\relax
    \let\do\@makeother\dospecials
    \catcode`\ =\active
    \catcode`\^^I=\active
    \xdef\minted@tmp{\scantokens{#1}}%
    \endgroup
    \let\FV@Line\minted@tmp
    \def\FV@SV@minted@tmp{%
      \FV@CodeLineNo=1\FV@StepLineNo
      \FV@Gobble
      \expandafter\FV@ProcessLine\expandafter{\FV@Line}}%
    \minted@langlinenoson
    \UseVerbatim{minted@tmp}%
    \minted@langlinenosoff
    \endgroup}}%
  {\newcommand{\mint@iii}[1]{%
    \endgroup
    \minted@writecmdcode{#1}%
    \minted@langlinenoson
    \minted@pygmentize{\minted@lang}%
    \minted@langlinenosoff
    \endgroup}}
```

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

```
\ifthenelse{\boolean{minted@draft}}%
  {\newenvironment{minted}[2][]
    {\VerbatimEnvironment
      \minted@configlang{#2}%
      \setkeys{minted@opt@cmd}{#1}%
      \minted@fvset
      \minted@langlinenoson
      \begin{Verbatim}}%
    {\end{Verbatim}%
      \minted@langlinenosoff}}%
  {\newenvironment{minted}[2][]
    {\VerbatimEnvironment
      \let\FVB@VerbatimOut\minted@FVB@VerbatimOut
      \let\FVE@VerbatimOut\minted@FVE@VerbatimOut
      \minted@configlang{#2}%
      \setkeys{minted@opt@cmd}{#1}%
      \minted@fvset
      \begin{VerbatimOut}[codes={\catcode`\^^I=12}]{\minted@jobname.pyg}}%
    {\end{VerbatimOut}%
        \minted@langlinenoson
        \minted@pygmentize{\minted@lang}%
        \minted@langlinenosoff}}
```

\inputminted Highlight an external source file.

```
\ifthenelse{\boolean{minted@draft}}%
  {\newcommand{\inputminted}[3][]{%
    \begingroup
    \minted@configlang{#2}%
    \setkeys{minted@opt@cmd}{#1}%
    \minted@fvset
    \VerbatimInput{#3}%
    \endgroup}}%
  {\newcommand{\inputminted}[3][]{%
    \begingroup
    \minted@configlang{#2}%
    \setkeys{minted@opt@cmd}{#1}%
    \minted@fvset
    \minted@pygmentize[#3]{#2}%
    \endgroup}}
```

9.11 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\newrobustcmd\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}}%
}
```

9.12 Float support

listing Define a new floating environment to use for floated listings. This is defined conditionally based on the `newfloat` package option.

```
\ifthenelse{\boolean{minted@newfloat}}%
 {\@ifundefined{minted@float@within}%
    {\DeclareFloatingEnvironment[fileext=lol,placement=h]{listing}}%
    {\def\minted@tmp#1{%
       \DeclareFloatingEnvironment[fileext=lol,placement=h, within=#1]{listing}}%
```

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

The following macros only apply when `listing` is created with the float package. When `listing` is created with newfloat, its properties should be modified using newfloat's `\SetupFloatingEnvironment`.

```
\ifminted@newfloat\else
```

`\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}}
```

Again, the preceding macros only apply when float is used to create listings, so we need to end the conditional.

```
\fi
```

9.13 Epilogue

Check whether LaTeX was invoked with `-shell-escape` option, set the default style, and make sure `pygmentize` exists. Checking for `pygmentize` must wait until the end of the preamble, in case it is specified via `\MintedPygmentize` (which would typically be after the package is loaded).

```
\AtEndOfPackage{%
  \ifthenelse{\boolean{minted@draft}}%
   {}%
   {%
```

```
    \ifthenelse{\boolean{minted@frozencache}}{}{%
      \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}%
   }%
}
\AtEndPreamble{%
  \ifthenelse{\boolean{minted@draft}}%
   {}%
   {%
    \ifthenelse{\boolean{minted@frozencache}}{}{%
      \TestAppExists{\MintedPygmentize}%
      \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}%
  }%
}
```

9.14 Final cleanup

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

```
\AfterEndDocument{%
  \ifthenelse{\boolean{minted@draft}}%
   {}%
   {\ifthenelse{\boolean{minted@frozencache}}%
     {}
     {\ifx\XeTeXinterchartoks\minted@undefined
      \else
        \DeleteFile[\minted@outputdir]{\minted@jobname.mintedcmd}%
        \DeleteFile[\minted@outputdir]{\minted@jobname.mintedmd5}%
      \fi
      \DeleteFile[\minted@outputdir]{\minted@jobname.pyg}%
      \DeleteFile[\minted@outputdir]{\minted@jobname.out.pyg}%
     }%
   }%
}
```

10 Implementation of compatibility package

minted version 2 is designed to be completely compatible with version 1.7. All of the same options and commands still exist. As far as most users are concerned, the only difference should be the new commands and options.

However, minted 2 does require some additional packages compared to minted 1.7. More importantly, since minted 2 has almost completely new internal code, user code that accessed the internals of 1.7 will generally not work with 2.0, at least not without some modification. For these reasons, a copy of minted 1.7 is supplied as the package minted1. This is intended *only* for compatibility cases when using the current version is too inconvenient.

The code in minted1 is an exact copy of minted version 1.7, except for two things: (1) the package has been renamed, and (2) code has been added that allows minted1 to act as (impersonate) minted, so that it can cooperate with other packages that require minted to be loaded.[9] When minted1 is used, it must be loaded *before* any other packages that would require minted.

All modifications to the original minted 1.7 source are indicated with comments. All original code that has been replaced has been commented out rather than deleted. Any future modifications of minted1 should *only* be for the purpose of allowing it to serve better as a drop-in compatibility substitute for the current release of minted.

```
\NeedsTeXFormat{LaTeX2e}
%%%% Begin minted1 modification
%%\ProvidesPackage{minted}[2011/09/17 v1.7 Yet another Pygments shim for LaTeX]
\ProvidesPackage{minted1}[2015/01/31 v1.0 minted 1.7 compatibility package]
%%%% End minted1 modification
\RequirePackage{keyval}
\RequirePackage{fancyvrb}
\RequirePackage{xcolor}
\RequirePackage{float}
\RequirePackage{ifthen}
%%%% Begin minted1 modification
\newboolean{mintedone@mintedloaded}
\@ifpackageloaded{minted}%
 {\setboolean{mintedone@mintedloaded}{true}%
  \PackageError{minted1}{The package "minted1" may not be loaded after
       ^^J"minted" has already been loaded--load "minted1" only for "minted"
       ^^Jversion 1.7 compatibility}%
   {Load "minted1" only when "minted" version 1.7 compatibility is required}}%
 {}
\ifmintedone@mintedloaded\else
\@namedef{ver@minted.sty}{2011/09/17 v1.7 Yet another Pygments shim for LaTeX}
```

[9]The approach used for doing this is described at http://tex.stackexchange.com/a/39418/10742.

```
\expandafter\let\expandafter\minted@tmp\csname opt@minted1.sty\endcsname
\expandafter\let\csname opt@minted.sty\endcsname\minted@tmp
\let\minted@tmp\relax
%%%% End minted1 modification
\RequirePackage{calc}
\RequirePackage{ifplatform}
\DeclareOption{chapter}{\def\minted@float@within{chapter}}
\DeclareOption{section}{\def\minted@float@within{section}}
\ProcessOptions\relax
\ifwindows
  \providecommand\DeleteFile[1]{\immediate\write18{del #1}}
\else
  \providecommand\DeleteFile[1]{\immediate\write18{rm #1}}
\fi
\newboolean{AppExists}
\newcommand\TestAppExists[1]{
  \ifwindows
    \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>>\jobnam
    \newread\@appexistsfile
    \immediate\openin\@appexistsfile\jobname.aex
    \expandafter\def\expandafter\@tmp@cr\expandafter{\the\endlinechar}
    \endlinechar=-1\relax
    \readline\@appexistsfile to \@apppathifexists
    \endlinechar=\@tmp@cr
    \ifthenelse{\equal{\@apppathifexists}{}}
     {\AppExistsfalse}
     {\AppExiststrue}
    \immediate\closein\@appexistsfile
    \DeleteFile{\jobname.aex}
\immediate\typeout{file deleted}
  \else
    \immediate\write18{which #1 && touch \jobname.aex}
    \IfFileExists{\jobname.aex}
     {\AppExiststrue
      \DeleteFile{\jobname.aex}}
     {\AppExistsfalse}
  \fi}
\newcommand\minted@resetoptions{}
\newcommand\minted@defopt[1]{
  \expandafter\def\expandafter\minted@resetoptions\expandafter{%
    \minted@resetoptions
    \@namedef{minted@opt@#1}{}}}
\newcommand\minted@opt[1]{
  \expandafter\detokenize%
    \expandafter\expandafter\expandafter{\csname minted@opt@#1\endcsname}}
\newcommand\minted@define@opt[3][]{
  \minted@defopt{#2}
  \ifthenelse{\equal{#1}{}}{
    \define@key{minted@opt}{#2}{\@namedef{minted@opt@#2}{#3}}}
   {\define@key{minted@opt}{#2}[#1]{\@namedef{minted@opt@#2}{#3}}}}
\newcommand\minted@define@switch[3][]{
  \minted@defopt{#2}
  \define@booleankey{minted@opt}{#2}
   {\@namedef{minted@opt@#2}{#3}}
   {\@namedef{minted@opt@#2}{#1}}}
\minted@defopt{extra}
\newcommand\minted@define@extra[1]{
  \define@key{minted@opt}{#1}{
    \expandafter\def\expandafter\minted@opt@extra\expandafter{%
      \minted@opt@extra,#1=##1}}}
\newcommand\minted@define@extra@switch[1]{
  \define@booleankey{minted@opt}{#1}
   {\expandafter\def\expandafter\minted@opt@extra\expandafter{%
      \minted@opt@extra,#1}}
   {\expandafter\def\expandafter\minted@opt@extra\expandafter{%
      \minted@opt@extra,#1=false}}}
\minted@define@switch{texcl}{-P texcomments}
\minted@define@switch{mathescape}{-P mathescape}
\minted@define@switch{linenos}{-P linenos}
\minted@define@switch{startinline}{-P startinline}
\minted@define@switch[-P funcnamehighlighting=False]%
  {funcnamehighlighting}{-P funcnamehighlighting}
\minted@define@opt{gobble}{-F gobble:n=#1}
\minted@define@opt{bgcolor}{#1}
\minted@define@extra{frame}
\minted@define@extra{framesep}
\minted@define@extra{framerule}
\minted@define@extra{rulecolor}
\minted@define@extra{numbersep}
\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}
\newsavebox{\minted@bgbox}
\newenvironment{minted@colorbg}[1]{
  \def\minted@bgcol{#1}
  \noindent
  \begin{lrbox}{\minted@bgbox}
  \begin{minipage}{\linewidth-2\fboxsep}}
 {\end{minipage}
  \end{lrbox}%
  \colorbox{\minted@bgcol}{\usebox{\minted@bgbox}}}
\newwrite\minted@code
\newcommand\minted@savecode[1]{
  \immediate\openout\minted@code\jobname.pyg
  \immediate\write\minted@code{#1}
  \immediate\closeout\minted@code}
\newcommand\minted@pygmentize[2][\jobname.pyg]{
  \def\minted@cmd{pygmentize -l #2 -f latex -F tokenmerge
    \minted@opt{gobble} \minted@opt{texcl} \minted@opt{mathescape}
    \minted@opt{startinline} \minted@opt{funcnamehighlighting}
    \minted@opt{linenos} -P "verboptions=\minted@opt{extra}"
    -o \jobname.out.pyg #1}
  \immediate\write18{\minted@cmd}
  % For debugging, uncomment:
  %\immediate\typeout{\minted@cmd}
  \ifthenelse{\equal{\minted@opt@bgcolor}{}}
   {}
   {\begin{minted@colorbg}{\minted@opt@bgcolor}}
  \input{\jobname.out.pyg}
  \ifthenelse{\equal{\minted@opt@bgcolor}{}}
   {}
   {\end{minted@colorbg}}
  \DeleteFile{\jobname.out.pyg}}
\newcommand\minted@usedefaultstyle{\usemintedstyle{default}}
\newcommand\usemintedstyle[1]{
  \renewcommand\minted@usedefaultstyle{}
  \immediate\write18{pygmentize -S #1 -f latex > \jobname.pyg}
  \input{\jobname.pyg}}
\newcommand\mint[3][]{
  \DefineShortVerb{#3}
  \minted@resetoptions
  \setkeys{minted@opt}{#1}
  \SaveVerb[aftersave={
    \UndefineShortVerb{#3}
    \minted@savecode{\FV@SV@minted@verb}
    \minted@pygmentize{#2}
    \DeleteFile{\jobname.pyg}}]{minted@verb}#3}
\newcommand\minted@proglang[1]{}
\newenvironment{minted}[2][]
 {\VerbatimEnvironment
```

```
  \renewcommand{\minted@proglang}[1]{#2}
  \minted@resetoptions
  \setkeys{minted@opt}{#1}
  \begin{VerbatimOut}[codes={\catcode`\^^I=12}]{\jobname.pyg}}%
 {\end{VerbatimOut}
  \minted@pygmentize{\minted@proglang{}}
  \DeleteFile{\jobname.pyg}}
\newcommand\inputminted[3][]{
  \minted@resetoptions
  \setkeys{minted@opt}{#1}
  \minted@pygmentize[#3]{#2}}
\newcommand\newminted[3][]{
  \ifthenelse{\equal{#1}{}}
   {\def\minted@envname{#2code}}
   {\def\minted@envname{#1}}
  \newenvironment{\minted@envname}
   {\VerbatimEnvironment\begin{minted}[#3]{#2}}
   {\end{minted}}
  \newenvironment{\minted@envname *}[1]
   {\VerbatimEnvironment\begin{minted}[#3,##1]{#2}}
   {\end{minted}}}
\newcommand\newmint[3][]{
  \ifthenelse{\equal{#1}{}}
   {\def\minted@shortname{#2}}
   {\def\minted@shortname{#1}}
  \expandafter\newcommand\csname\minted@shortname\endcsname[2][]{
    \mint[#3,##1]{#2}##2}}
\newcommand\newmintedfile[3][]{
  \ifthenelse{\equal{#1}{}}
   {\def\minted@shortname{#2file}}
   {\def\minted@shortname{#1}}
  \expandafter\newcommand\csname\minted@shortname\endcsname[2][]{
    \inputminted[#3,##1]{#2}{##2}}}
\@ifundefined{minted@float@within}
 {\newfloat{listing}{h}{lol}}
 {\newfloat{listing}{h}{lol}[\minted@float@within]}
\newcommand\listingscaption{Listing}
\floatname{listing}{\listingscaption}
\newcommand\listoflistingscaption{List of listings}
\providecommand\listoflistings{\listof{listing}{\listoflistingscaption}}
\AtBeginDocument{
  \minted@usedefaultstyle}
\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}
%%%% Begin minted1 modification
\fi
%%%% End minted1 modification
```