NAME

Text::Wrap - line wrapping to form simple paragraphs

SYNOPSIS

Example 1

```
use Text::Wrap;

$initial_tab = "\t"; # Tab before first line
$subsequent_tab = ""; # All other lines flush left

print wrap($initial_tab, $subsequent_tab, @text);
print fill($initial_tab, $subsequent_tab, @text);

$lines = wrap($initial_tab, $subsequent_tab, @text);

@paragraphs = fill($initial_tab, $subsequent_tab, @text);
```

Example 2

```
use Text::Wrap qw(wrap $columns $huge);

$columns = 132;  # Wrap at 132 characters
$huge = 'die';
$huge = 'wrap';
$huge = 'overflow';
```

Example 3

```
use Text::Wrap;

$Text::Wrap::columns = 72;
print wrap('', '', @text);
```

DESCRIPTION

`Text::Wrap::wrap()` is a very simple paragraph formatter. It formats a single paragraph at a time by breaking lines at word boundaries. Indentation is controlled for the first line (`$initial_tab`) and all subsequent lines (`$subsequent_tab`) independently. Please note: `$initial_tab` and `$subsequent_tab` are the literal strings that will be used: it is unlikely you would want to pass in a number.

`Text::Wrap::fill()` is a simple multi-paragraph formatter. It formats each paragraph separately and then joins them together when it's done. It will destroy any whitespace in the original text. It breaks text into paragraphs by looking for whitespace after a newline. In other respects, it acts like wrap().

`wrap()` compresses trailing whitespace into one newline, and `fill()` deletes all trailing whitespace.

Both `wrap()` and `fill()` return a single string.

Unlike the old Unix fmt(1) utility, this module correctly accounts for any Unicode combining characters (such as diacriticals) that may occur in each line for both expansion and unexpansion. These are overstrike characters that do not increment the logical position. Make sure you have the appropriate Unicode settings enabled.

OVERRIDES

`Text::Wrap::wrap()` has a number of variables that control its behavior. Because other modules might be using `Text::Wrap::wrap()` it is suggested that you leave these variables alone! If you can't do that, then use `local($Text::Wrap::VARIABLE) = YOURVALUE` when you change the values so that the original value is restored. This `local()` trick will not work if you import the variable into your own namespace.

Lines are wrapped at `$Text::Wrap::columns` columns (default value: 76). `$Text::Wrap::columns` should be set to the full width of your output device. In fact, every resulting line will have length of no more than `$columns - 1`.

It is possible to control which characters terminate words by modifying `$Text::Wrap::break`. Set this to a string such as `'[\s:]'` (to break before spaces or colons) or a pre-compiled regexp such as `qr/[\s']/` (to break before spaces or apostrophes). The default is simply `'\s'`; that is, words are terminated by spaces. (This means, among other things, that trailing punctuation such as full stops or commas stay with the word they are "attached" to.) Setting `$Text::Wrap::break` to a regular expression that doesn't eat any characters (perhaps just a forward look-ahead assertion) will cause warnings.

Beginner note: In example 2, above `$columns` is imported into the local namespace, and set locally. In example 3, `$Text::Wrap::columns` is set in its own namespace without importing it.

`Text::Wrap::wrap()` starts its work by expanding all the tabs in its input into spaces. The last thing it does it to turn spaces back into tabs. If you do not want tabs in your results, set `$Text::Wrap::unexpand` to a false value. Likewise if you do not want to use 8-character tabstops, set `$Text::Wrap::tabstop` to the number of characters you do want for your tabstops.

If you want to separate your lines with something other than `\n` then set `$Text::Wrap::separator` to your preference. This replaces all newlines with `$Text::Wrap::separator`. If you just want to preserve existing newlines but add new breaks with something else, set `$Text::Wrap::separator2` instead.

When words that are longer than `$columns` are encountered, they are broken up. `wrap()` adds a `"\n"` at column `$columns`. This behavior can be overridden by setting `$huge` to 'die' or to 'overflow'. When set to 'die', large words will cause `die()` to be called. When set to 'overflow', large words will be left intact.

Historical notes: 'die' used to be the default value of `$huge`. Now, 'wrap' is the default value.

EXAMPLES

Code:

```
print wrap("\t","",<<END);
This is a bit of text that forms
a normal book-style indented paragraph
END
```

Result:

```
" This is a bit of text that forms
a normal book-style indented paragraph
"
```

Code:

```
$Text::Wrap::columns=20;
$Text::Wrap::separator="|";
print wrap("","","This is a bit of text that forms a normal book-style
```

paragraph");Result:

```
  "This is a bit of|text that forms a|normal book-style|paragraph"
```

SUBVERSION

This module comes in two flavors: one for modern perls (5.10 and above) and one for ancient obsolete perls. The version for modern perls has support for Unicode. The version for old perls does not. You can tell which version you have installed by looking at `$Text::Wrap::SUBVERSION`: it is `old` for obsolete perls and `modern` for current perls.

This man page is for the version for modern perls and so that's probably what you've got.

SEE ALSO

For correct handling of East Asian half- and full-width characters, see *Text::WrapI18N*. For more detailed controls: *Text::Format*.

AUTHOR

David Muir Sharnoff <cpan@dave.sharnoff.org> with help from Tim Pierce and many many others.

LICENSE