gendoc convert inline documentation in a script to HTML or PDF

Synopsis

```
gendoc [options] script
```

Options:

`-h,--help`	print this help and exit
`-H,--Help`	print full documentation via less and exit
`-V,--version`	print version and exit
`-g,--gray`	inline code gets gray background, with normal spaces
`-p,--pdf`	generate a PDF file (in current working directory) instead of HTML
`-v,--verbose`	print some intermediate messages
`-d,--debug`	set debug flag: don't delete some intermediate files for inspection

The script must be in your PATH, and in its directory there must be either a subdirectory `doc` generated by gendoc, or no such subdirectory at all.

Description

gendoc looks for simplified documentation in any script, placed between a line containing nothing but the string `<<'DOC'` and another line, containing nothing but the string `DOC` and converts it to an HTML document (the default) or, if the `--pdf,-p` option was used, to a PDF document, *via* LuaLaTeX. The output is written to the directory `doc` under the directory where the script is located.

Bars (|) frequently occur in this type of documentation, because they are used to typeset literal text in lightblue typewriter font. You can typeset a single bar, like the one in the beginning of this sentence, by putting it between parentheses.

The simplification mentioned consists of the following:

- For an HTML document, the `<body>` and `</body>` statements and everythingoutside this pair is automatically generated.
- Similarly, for a PDF document, the `\begin{document}` and `\end{document}` statements and everything outside this pair is automatically generated.
- The name of the script will automatically be printed in bold face, it wil be linked to `../<scriptname>`, and characters will be escaped if needed.
- A line starting with a single `=` is typeset as a heading.
- The first heading should be of the form: `=␣scriptname␣-␣description`; it will be typeset larger than other headers.
- A line starting with `-␣` (a hyphen and one space) is the first item of a bulleted list, which ends when a line without a starting `-␣` or `␣␣` (two spaces) is met. Lines starting with a tab or with more than 2 whitespace characters are typeset as code, like in the next item:
- Outside bulleted lists, lines starting with whitespace are typeset verbatim, like this:

  ```
  #!/bin/bash
  # Start of my script
  ```

  In such lines, pairs of vertical bars (|), asterisks (*) and slashes (/) are typeset literally, *i.e.*, they will not influence the typeface of the surrounded text. However, you *can* make verbatim text bold, italic or colored with the `XXX{...}` sequences explained below. However, this will only work for HTML output, as such sequences will be removed (with a warning) when a PDF document is produced.
- Lines containing a tab character are typeset as two-column tables, with left-aligned cells. The first column will be typeset verbatim. Also, lines in second column will be typeset verbatim if they start with at least one space. If the first line of the table has an empty second field, i.e. ends with a tab, all lines are supposed to have only one non-empty field and non-empty first fields are typeset over two columns, while empty first fields wil have a small fixed width: 10% for HTML output, 10mm for PDF output.

Some special commands are defined in order to keep markup as simple as possible. Currently these are:

`|...|`	prints the `...` as code in light blue typewriter font. If you need to typeset a bar (|), enclose it in parentheses; those are also typeset.
`B{...}`	prints its argument **bold**
`*...*`	does the same; if you need asterisks, offer them in verbatime text, either between vertical bars or space-prefixed lines.
`I{...}`	prints its argument *italic*
`/.../`	does the same; if you need slashes, offer them in verbatime text, either between vertical bars or space-prefixed lines.

`U{...}{...}`	prints second argument and links it to url in first argument. If the first argument contains an @ character, `mailto:` will automatically be added, if it's not already there; similarly, if there is no `http://` or `https://`, `http://` will automatically be inserted. So you can simply write:
	`U{bc@def.com}{mail me} and`
	`U{www.google.com}{Google}`

`Red{...}`	prints its argument in red color
`Green{...}`	prints its argument in green color
`Blue{...}`	prints its argument in blue color

Version and type

The script's version are displayed in the documentation. The type is taken from the shebang line. The version is captured from the script by looking for a line starting with `Version`, `version` or `$version`, followed by `=n.mx`, where the `=` may be surrounded by whitespace, `n` and `m` are one or more digits, and `x` stands for zero or more lowercase letters.

Recreate all docs

If for some reason a `doc` subdirectory needs to be regenerated, then cd to the directory above it and run:

```
for i in $(grep -d skip -l "^<<'DOC'" *); do gendoc ./$i; done
```

Author and copyright

`Author`	Wybo Dekker
`Email`	wybo@dekkerdocumenten.nl
`License`	Released under the GNU General Public License

Functions used:

verb

`parameters:`	1+: strings to print
`description:`	Prints the argument to stderr, but only if verbose=true
`globals set:`	-
`globals used:`	verbose
`returns:`	0

info

`parameters:`	1: the script (in PATH) to inspect
`description:`	Find script's type, version, short description. see *scriptinfo* for more information.
`globals set:`	scriptversion type short
`globals used:`	-
`returns:`	0

indx

parameters:	-
description:	make new index in doc directory
globals set:	(via *scriptinfo*) type short scriptversion
globals used:	type short
returns:	0

fnd

parameters:	1: name of the script to be found
description:	Find the script and set some globals, telling the directory where the script is in, the basename of the script and its type. The script must be in your PATH, and its directory must contain a subdirectory named doc. If there is no such directory, it is (optionally) created.
globals set:	dir base type
globals used:	indexheader
returns:	1 on error, 0 otherwise.

lookfor

parameters:	1: line number of from where a string is searched 2: the string to be searched
description:	Remembers, for an eventual error message, from which starting line string is being searched, and the string we are looking for.
globals set:	startingat lookingfor
globals used:	-
returns:	0

do_presets

parameters:	1: the output extension, pdf or html
description:	Preset many global variables, depending on the output extension (html or pdf)
returns:	-
globals set:	too many to report here
globals used:	