Simplewp documentation

Contents

1 What is simplewp ?

Simplewp creates an entire static website from a set of simple text files. It is written in Python[1] and uses reStructuredText[2] to convert plain text to HTML[3].

The created website can contain a menu with links to the pdf, plain text and printable version of the current page, as well as some text describing where the user is in the site.

Simplewp supports website with multiple languages and can also include links to translated version of the same content in the website's menu.

You can also write math equations using the latex syntax[4].

[1] `http://python.org/`
[2] `http://docutils.sourceforge.net/rst.html`
[3] `http://en.wikipedia.org/wiki/HTML`
[4] `http://svasey.org/projects/old-stuff/svrst_en.html#the-math-extension`

2 Why reStructuredText ?

HTML is easily readable by machine (and by some humans), but it realy is cumbersome to write because of its redundancy. Plain text lets you focus on the content, forgetting about most of the technical and presentation details.

reStructuredText had all the features I wanted, plus a very handy extension mechanism to add math support. Furthermore, it is very readable in its plain text form, which is especially useful when you do not want to preview the text in HTML all the time.

3 How to use it ?

If you know reStructuredText[5], simplewp is really simple to use: create a directory hierarchy containing at least the following files and directories

- `global.conf`: contain some website configuration options
- `html/`: contains the html menu and footer files you might want to include on the top and bottom of each of your website's page
- `rst/`: contains the content of your website: plaintext `.rst` files, that will each become a page of your website
- `scripts/`: might contain some scripts to run after or before the rst files have been compiled. I personally use this to generate a robots.txt[6] file, but other uses are possible, of course
- `tocopy/`: contains all the files that you just want to copy into your website unchanged. This might include images, css stylesheets, and other static content.

Assume this hierarchy has been created inside the directory `website-source`

You can then create a build directory, where the content of your website will appear in its final form:

```
$ mkdir website-build
```

then just build your website:

```
$ wpcompile website-source website-build
```

then check out the result:

```
$ firefox website-build/somepage.html
```

etc...

To update your website, just change some of the content in the source and rebuilt it using the same command as before. Only the sources that have changed will be recompiled.

Of course, it might happen that someone visits your site must when it is being compiled. In that case he might be served some inconsistent content or get some error. To avoid that, you should use the `website-update` utility:

[5] `http://docutils.sourceforge.net/rst.html`
[6] `http://www.robotstxt.org/`

```
$ website-update --content-directory=. website-content website-build
```

This will create a `website-content` symbolic link, and will change it everytime the content changes. You should then serve the website in `website-content` to your users. Since a symbolic link change is atomic on most UNIX-based systems, a user that is served a page while the updating is done will either see the old site or the new site, but never a mix of both.

4 Program usage

4.1 wpcompile

`wpcompile` compiles a source hierarchy into a complete website. It takes two optionnal arguments.

- The source directory (by default the current directory)
- The build directory: where the complete website should be built (by default in `$sourcedir/build`).

Normally, `wpcompile` will only compile files that have changed since the last compilation. However, if you want everything to be recompiled for some reason, you can use the `--buildall` flag.

4.2 website-update

`website-update` maintains a symbolic link to your website's content. It is used to update your website in an atomic way. This is done by copying the content of your build directory to a temporary directory and updating the symbolic link to point to the latter.

The program takes exactly two arguments:

- The path to the symbolic link (it is not necessary that it already exists)
- The path to the website content to put in place (i.e the build directory)

The following options can be used:

- `--group=GROUPNAME`: the name of the group that will own the temporary directory and its content. The default is the current group.
- `--user=GROUPNAME`: the name of the user that will own the temporary directory and its content. The default is the current user.
- `--content-directory=DIR`: The directory in which to create the temporary directory. If this is not specified, the directory name of the directory pointed to by the symbolic link will be used. For example if the symbolic link points to `/home/www/tmp/website-update...abcdef`, the content directory will be `/home/www/tmp/`. This must be specified if the symbolic link has not been created.
- `--no-remove-old`: By default, the directory that the symbolic link pointed to before the update is removed. If you do not want this to be the case, use that option.
- `--to-copy=FILE`: Specify additional content (file or directory) to copy into the website. This is useful if something that was not part of the simplewp hierarchy is also part of the site (see the Trac documentation for more). This can be given multiple times.

5 The source hierarchy

Here are the details on the elements of the source hierarchy. Note that any element that is not described here will be ignored by the compiler and can therefore be used for other things safely.

5.1 global.conf

This contains options global to the whole website. The syntax of this file is that of an INI file[7]. An example file would be:

```
[DEFAULT]
root-url = http://example.com
```

Note that this file is optionnal. The `root-url` variable is the only variable that is taken into consideration. It should contain the url to the root of the website. This is used when generating the relative link in the PDF file.

5.2 rst/

This is the most important directory of all, as it should contain the content of your website. This can contain any number of `.rst` files, contained in any number of directories and subdirectories. Each of these file will be converted to html and pdf, and the resulting hierarchy will mirror that of rst files. For example if the `rst/` directory contains the following files:

```
index.rst
somepage.rst
other/
    otherpage.rst
    other2/
        secondpage.rst
```

then after compilation, the hierarchy will look like this in the build directory:

```
index.source
index.chtml
index.pdf
index.html
somepage.source
somepage.chtml
somepage.pdf
somepage.html
other/
otherpage.source
    otherpage.chtml
    otherpage.pdf
    otherpage.html
    other2/
```

[7] `http://en.wikipedia.org/wiki/INI_file`

```
secondpage.source
secondpage.chtml
secondpage.pdf
secondpage.html
```

The `.chtml` files contain only the html converted from the `rst` file (it is used as a printable version), whereas the `.html` files contain this and the header and footer. The `.source` files are the original `rst` source.

Note that you should setup your web server to give the proper mime type for these files (i.e text/plain for `.source` files and text/html for `.chtml`).

5.2.1 Naming

As seen before, all RST files must end in `.rst`. Additionnaly, you can add a language indication by making the part before the `.rst` end with the two-letter code of the language in which the document is written, prefixed by an underscore (_). For example, if your document is written in English, you could name it `document_en.rst`. In that case, `document_fr.rst` would be treated specially by `wpcompile` as the French translation of the document.

If no language is given, English is assumed. It is good practice to always give a language code, just in case a translation is done later.

If you do not want to have a very large RST file compiled to html, you can split it up into smaller parts: to do that, just add a number before the `.rst` part of the name. For example, the files `index_en.00.rst` and `index_en.10.rst` will be compiled into the single file `index_en.html`, with the content of `index_en.00.rst` followed by that of `index_en.10.rst`.

There is a special file named `head.rst`, that should not be used as a document name. In it, you should put code that you think should go on top of every rst file you compile. For example, I use it to enable the math role by default. If a file `head.rst` is in `$dir`, all the RST files below `$dir`, including those in subdirectories will have the content of that file prepended. However, if there is an `head.rst` file in `$dir/$subdir`, the files below `$dir/$subdir` will use that `head.rst` file.

5.2.2 File-specific configuration

What if you do not want a specific RST file to have the content of `head.rst` included ? What if you do not want it to be compiled to PDF ? In that case you have to create a configuration file for the specific RST file. You should name it `name-without-rst.conf`. For example, if the RST file is `index_en.rst`, the configuration file would be named `index_en.conf`. The syntax of the configuration file is that of a simple INI file[8]. The variables that are taken into consideration are

- `stylesheet`: specify an alternate css stylesheet for the html file
- `pdfopts`: additionnal options to pass to the pdf compiler
- `htmlopts`: additionnal options to pass to the html compiler
- `nopdf`: if specified, this means the file should not be compiled to pdf

[8] `http://en.wikipedia.org/wiki/INI_file`

- `alternatepdf`: if specified, this means the file should not be compiled to pdf, but that another PDF file, named `$basename.pdf` will be provided. The only difference with `nopdf` is that a link to the PDF version is still created. It is the user's responsibility to create the PDF file.
- `nohtml`: if specified, this means the file should not be compiled to html
- `title`: alternate title for the html file. By default, the top-level header of the RST file is used.
- `print-stylesheet`: css stylesheet used for the printable (chtml) version
- `nohead`: if specified, this means we should not happen the head.rst file to the source file
- `alternatecat`: give a list of space-separated files (path given relative to `html/`) to be put before and after the html version of the rst file.

5.3 html/

This contains the html files to be put before and after the html version of the rst file. They are usually used to include a menu and a footer to each page. You can put special variables in the html files, and they will be replaced by some useful content at compile-time.

5.3.1 Naming

Each html file should be named according to the following format:

```
$basename.$language.$relpos.$pos.html
```

Where

- `$basename` can be anything you like and serves to identify the file
- `$language` is the two character language code of the document, or `all` if it is to be used with any language.
- `$relpos` is either `bef` (before) or `aft` (after), the position of the html file relative to the RST content
- `$pos` is a number indicating the order in which the html files must be concatenated. For example, if there is `footer.en.aft.00.html` and and `end.en.aft.10.html`, `end` will come after `footer`.

5.3.2 Variables

If you include a special string that begins and ends with three underscores (`___`), it will be considered a variable and be replaced at compile time by its value. Here is a list of all the variables

- `LANG` contains the two-character code of the file's language or `en` if unknown.
- `TITLE` contains the page's title

- `SYLESHEET` contains the path to the css stylesheet. By default, it is `css/default.css` (relative to the root directory, but the path is changed depending on the position of the html file in the hierarchy
- `SITE_TREE` return some html indicating where the page is in the site hierarchy. For example, from a page named `projects_en.html` located at the root of the hierarchy, this will look like this:

```
<span class="site-tree-text">
  <a href="../index_en.html" class="site-tree-link">
    home
  </a>
  /
  <span class="site-tree-nolink" >
    projects
  </span>
</span>
```

- `TRANSLATION_LINKS` If the page is available in more than one language, print a small menu with a link to the alternate version of the page. For example, if the page is available in English and French, and the currently viewed version is English, this will return something like this:

```
<p>
  <span class="translation-nolink" >
    EN
  </span>
  |
  <a href="test.fr.html" class="translation-link">
    FR
  </a>
</p>
```

- `PAGE_SOURCE` Returns the path to the plaintext version of the page.
- `PDF_VERSION_LINK` If no pdf version exists, return nothing. Otherwise, return a link to the pdf version of the page, preceeded by a " | ". For example:

```
| <a href="test_en.pdf">PDF</a>
```

- `PAGE_COMPILED_HTML` Return the path to the printable version of the page (`.chtml` file).
- `ROOT_DIR` Return the relative path to the hierarchy root directory.
- `YEAR` Return the current year as a 4 digit number
- `MONTH` Return the current month as a 2 digits number
- `MDAY` Return the current day of the month as a 2 digits number

5.4 scripts/

Scripts in that directory whose name begins with `precompile.` will be run before the rst files are compiled to html, scripts whose name begins with `postcompile.` will be run after the files are compiled to html. As argument, those scripst will be passed the path to the root source directory and the path to the build directory. Here is one of my postcompile script, that generates a robots.txt[9] file:

```
#!/usr/bin/env python

### Generate the robots.txt file. The generated file is compatible with the
### document published at http://www.robotstxt.org/orig.html (i.e no wildcards
### or other Allow statement

import sys

from os.path import join
from os import getcwd

from simplewp.robotstxt import excludeAlternative

sourceDir = sys.argv[1]
buildDir = sys.argv[2]

rstDir = join (sourceDir, 'rst')
outputFile = join (buildDir, 'robots.txt')

disallowAlt = excludeAlternative (rstDir)

with open (outputFile, 'w') as outputStream:
    outputStream.write ('User-agent: *\n')
    outputStream.writelines (disallowAlt)
```

The `simplewp.robotstxt.excludeAlternative` function generates a list of strings containing each line of the robots.txt file excluding each non-html version of the same page. This is done to avoid duplicate content indexing, as this might hamper the site's rating[10].

5.5 tocopy/

Files in that directory are just copied as is in the build hierarchy. So for example, if you have a `css/` directory in `tocopy`, and your build directory is `/home/www/build`, then once the site is built, there will be a `/home/www/build/css/` directory mirroring the content of `css/`

Note that you should really put a `css` directory in `tocopy`: by default, the path to the CSS stylesheet of the `.html` files is `css/default.css` in the build directory. It is `css/print.css` for the `.chtml` files. This means you should create those files and use them as stylesheets.

[9] `http://www.robotstxt.org/`

[10] `http://support.google.com/webmasters/bin/answer.py?hl=en&answer=66359`

6 RST format: improvements and limitations

I use my own programs (`svrst`) to build `rst` files. Even though they heavily use the ones given with docutils, there are some differences that you should take into account.

The main improvement is that you can write math. The main limitation is that some of the syntax is forbidden if you want to be able to convert into pdf, as I made the pdfs look nicer than originally at the price of some heavy hacking. See the svrst page[11] for more.

Note that the conversion to PDF works, but somewhat does not look as good as one would want it to. You can always add latex options into the `/etc/simplewp/stylesheet.tex` file to try to work around that.

[11] `http://svasey.org/projects/old-stuff/svrst_en.html`