Fronde Documentation

Étienne Deparis

October 15, 2024

Contents

This documentation assume you are using fronde version **0.6.0**.

1 Introduction

1.1 Summary

Fronde is a static website generator for Org, written in Ruby and Emacs Lisp (elisp). It enables you to concentrate on your writing, and manage all the needed steps to build and publish your website.

You can refer to the read me file for a quick list of the main features.

1.2 Nomenclature

This project name is fronde and will always be written normally though this document to refer to this project as a whole. As such, it will also be capitalized when used as the first word of a sentence. But when it is written as `fronde`, it will always refer to the provided program you can call in your terminal.

This document uses the short name Emacs to refer to the GNU Emacs editor or capabilities.

1.3 Frequently Asked Questions

1.3.1 Why do you think we need another static website generator?

My main reason to write fronde, is that I was looking for a static website generator, which provides (good) support for Org as a source format. But they somehow all lack of it. The second main reason I wrote this, instead of dealing with a pure Org publish config, was my need to easily amend the generated HTML. I'm not an elisp expert and I would like to have some generic solution for all the tiny websites I'm maintaining. In short, the main feature of this project is surely its templates support.

1.3.2 Where does this name come from?

A *fronde* is the french word for a sling. This name was chosen as a tribute to the french newspaper « La Fronde », created by Marguerite Durand in France in 1897. This was the first all-women written and edited newspaper in the world. Marguerite Durand said about it in one of her editorial[1] that her newspaper was « comme les autres journaux… pas plus amusant » (which may be translated as "just like other newspapers… as boring").

As such, fronde is like all other static website generator… as boring. However, like its illustrious reference, fronde wishes to help empowering people, offering them a tool to express themselves on the Web.

1.3.3 Opinionated you say?

Fronde is built with some strong opinion about what a static website generator should be, and more specifically, what should it be for Org.

- Org is a powerful tool that enables you to publish your writings in a wide range of formats. Its markup is easy to learn and as readable and as convenient as Markdown's. That's why fronde will never support any other markup language.
- However Org HTML export should be more easily expandable. That's why fronde gives you the templates feature and introduces new formatting tags for the preamble and postamble sections.
- Org writers should already be Emacs users, that's why fronde will directly call Emacs to convert your Org files to HTML. There is no need to use Pandoc or org-ruby if you already have Emacs.
- You may want to author a wide variety of things, and managing all these projects inside your Emacs configuration may be a nightmare. You also may want to apply different settings for different projects. That's why fronde is totally independent from your Emacs settings and enables you to organize your projects as you wish.

1.3.4 What will come next?

We are already aware of some limitations we would like to remove:

[1] Marguerite Durand. « En cinq ans ». La Fronde, n° 1832. 15th December 1902. https://gallica.bnf.fr/ark:/12148/bpt6k67059454.item

TODO Allow themes to provide configuration options Themes should be able to declare a set of default `templates` or specific Org export options

TODO Better protected upgrade For example, Rakefile should only be overwritten after upgrade of the gem.

TODO Create a themes portfolio

1.4 Source code

The canonical repository is located at `https://git.umaneti.net/fronde`. A mirror also exists on Github.

2 Getting started

2.1 Installation

Fronde is a Ruby gem and thus you need a working Ruby environment to use it. Fronde requires at least ruby 2.7 to work (though this version is no more maintained, you should really move to ruby 3.1 as soon as possible).

You may already have Ruby installed on your computer. You can check inside a terminal emulator by typing: `ruby -v`. You also need the rubygems package manager, which may also be already installed on your system. You can check it too with the following command: `gem -v`.

2.1.1 Install ruby with your system package manager

See the official ruby lang website to find out about options when installing a ruby environment on your system.

Some operating systems already package the right ruby version to use:

Archlinux-like On Archlinux, Manjaro or Parabola, you just have to install the packages `rubygems` (which will install `ruby` as a dependency):

```
sudo pacman -S ruby rubygems
```

Listing 1: Ruby installation procedure on Archlinux-like system

Debian-like On Debian, Ubuntu, Mint... you just have to install the package `ruby` (which will install `ruby-rubygems` as a dependency):

```
sudo apt install ruby
```

Listing 2: Ruby installation procedure on Debian-like system

Others Ruby seems to be well packaged for Fedora, Red Hat or OpenMandriva, thus it should not be a problem for you to install it.

2.1.2 Install ruby with RVM

RVM allows you to install different ruby versions on the same machine. It could be interesting when you are already a ruby developer.

We recommend that you use a dedicated gemset for fronde, to avoid polluting your other projects.

```
rvm get latest
rvm install ruby-3.2.2
rvm use ruby-3.2.2@fronde --create
```

Listing 3: Ruby installation procedure with RVM

2.1.3 Install Fronde

As soon as you have a working ruby environment, you just need to install it as any other gem:

```
gem install fronde
```

Listing 4: Fronde gem installation procedure

2.1.4 Build Fronde from sources

You may also want to install a development version of fronde, directly from its source code repository. We won't describe this procedure in detail as it should only be used by developers or advanced users. But still, here are the commands you can used to build fronde from the sources.

Then, you can install fronde from this new locally built gem file (obviously, you must adapt the following command with the fronde version number you just built):

```
git clone https://git.umaneti.net/fronde
cd fronde
gem install bundler
bundle install
gem build fronde.gemspec
```

Listing 5: Build sources procedure

```
gem install fronde-x.y.z.gem
```

Listing 6: Fronde installation procedure from local gem file

2.2 Creation of a website

A fronde project is essentially just a folder containing the configuration of your website and a subfolder containing the org files to publish. fronde requires a local installation of org-mode to work, and will generate various configuration files you are not expected to modify. To set up all these things, you must use the `fronde init` command.

Thus, to create a new fronde website, you must create an empty folder and move to this new folder in your terminal:

```
mkdir yourproject
cd yourproject
```

Listing 7: Create a fronde project directory

Now, you can run the `fronde init` command with some arguments:

The following arguments are allowed:

-a (–author) Set up the default author name (see the `author` config key)

-l (–lang) Set up the default lang of your website (see the `lang` config key)

-t (–title) Set up the title of your first page

At the end of the init process, fronde will automatically open Emacs on the

```
fronde init -t "My brand new website"
```

Listing 8: Initialize a fronde project

first page of your website. You can directly write in it or close it and come back to it later. This file is stored in the default `src` source subfolder as `src/index.org`.

Your project now consist of the following file hierarchy:

```
yourproject/
├── config.yml      # main config file
├── lib/            # required dependencies
├── public_html/    # output folder
|   └── assets/
├── Rakefile        # fronde-generated ruby related file
├── src/
|   └── index.org   # your first website page
└── var/            # variable files used during operation
```

Fronde relies a lot on files stored in the `lib` and `var` folders. You should never try to remove them by yourself.

2.3 Creation or edition of a page

To create a new page for your website, or to edit an already existing one, you are totally free to use the tool you want. Pages are just regular Org files, without any specific modifications. If you already have a bunch of them, you can use them without any changes.

By default, without any other configuration options, the website pages must be stored in a folder named `src` at the root of your project. However you can configure any other sources folder you want, even one that is not in your project directory.

In parallel to your regular Org workflow, fronde provides command to help you create or edit pages for your website: the `fronde open` command.

The most simple use case is to call `fronde open` with a file path and it will open that file in your default `EDITOR` (which should be Emacs).

```
fronde open src/index.org
```

Listing 9: Open a page with the fronde open command

When creating a new page, this command accepts the following arguments:

-a (–author) Set the author name of the page

-l (–lang) Set the language of the page

-t (–title) Set the title of the page

If you use the `--title` argument, instead of giving a full file path argument, you can just give the folder path where you want to save the new file, and the command will create the document with a web-ready name.

```
fronde open -a Alice -t "My new page's shiny, isn't it?" src
ls src
> index.org  my-new-page-s-shiny-isn-t-it.org
```

Listing 10: Creation of a new page with the title argument

And now `src/my-new-page-s-shiny-isn-t-it.org` contains:

```
#+title: My new page's shiny, isn't it?
#+date: <2020-11-12 Thu. 11:25:58>
#+author: Alice
#+language: en
```

2.4 Building the website

Once you have written some content, you can convert your org files to HTML with the `fronde build` command.

```
fronde build
```

Listing 11: Build a fronde project

Because this building process makes a direct use of the org-mode publishing feature, it will rebuild only files changed since the last command invocation. If you want to force a full rebuild, you can pass the `--force` argument to the command.

```
fronde build -f
```

To review what you just built, you can use the `fronde preview` command, which will start a local webserver and open your default web browser on the home page of your project.

```
fronde preview
```

Listing 12: Build a fronde project

2.5 Publishing the website

To publish your website, just use the following command:

```
fronde publish
```

Listing 13: Publish a fronde project

You can give the `--verbose` argument to display more detail on the operation.

```
fronde publish -v
```

Listing 14: Publish verbosely a fronde project

The publication target must be defined using one of the two settings gemini_remote or html_remote. If none of those setting is set, the `publish` command will do nothing.

3 Advanced usage

3.1 Config File

All the fronde configuration is stored in one YAML file named `config.yml` and stored at the root of your project. If the file does not exist a default one will be created the first time you run any `fronde` command.

The following explains all possible configuration options for fronde, which may be put in file named `config.yml` at the root of your static website project. The options are listed in alphabetical order. However, they can be put in any order in the `config.yml` file.

3.1.1 `author`

The `author` option stores the default author name of your org files. This value is used when you create a new file. **It is not used by the regular Org mode publishing process**. You must be sure that your org files contains an `#+author:` metadata field before publishing them.

This value is used in the generation of the blog index page and the main atom feed.

```
---
author: Alice Doe
```

Listing 15: Example of an author setting

The default `author` setting is your user name on your current computer session.

3.1.2 `domain`

The domain name pointing to where your static website is expected to be hosted. This value is used to generate absolute path to your files on your expected host name (for example in the Atom feeds).

Its value is expected to **not** end with a slash. That is to say `https://example.com` is **valid**, whereas `https://example.com/` is **not**.

By default, your static website is expected to be hosted at the root of your domain name (like `https://example.com/index.html`). If it is not the case and your fronde static website is hosted in a subfolder of your main website, you must add this subfolder to the `domain` value. Thus, `https://example.com/example` or `https://example.com/complex/example` are **valid** values too.

```
---
domain: https://alice.doe.name
```

Listing 16: Example of a domain setting

The default `domain` value is an empty string.

3.1.3 `gemini_public_folder` or `html_public_folder`

The `gemini_public_folder` or `html_public_folder` option stores the path to the folder, which will respectively contains the generated gemtext or HTML files, ready for Gemini or Web publication.

The default `gemini_public_folder` value is `public_gmi` (at the root of your project).

The default `html_public_folder` value is `public_html` (at the root of your project).

3.1.4 `gemini_remote` or `html_remote`

The `gemini_remote` or `html_remote` option contains the destination of your capsule or website. This destination string define whether the Neocities or the default rsync backend must be used, and with what parameters.

There is no default value for `gemini_remote` or `html_remote`. Publication of your capsule will be skipped if there is no value for `gemini_remote`. The same thing for your website, if there is no value for `html_remote`, the publication of your website will be skipped.

Rsync backend If you plan to use rsync to publish your website, you just have to give the destination target, as you would have done on the command line. For example `user@domain:/var/www/mywebsite/`

The trailing slash is mandatory.

Fronde will directly call Rsync in a synchronous process, thus if a password must be entered to either unlock an SSH key, or because your are trying to connect with a password, you will be able to manually enter it as usual.

Neocities backend If you plan to publish your website on a Website offering the same API than Neocities, you can use the following value for this setting: `neocities:<website_name>@<server_name>`

`<website_name>` is your Neocities website name (the username you use to log in)

`<server_name>` is the Neocities server you want to publish on (Neocities is a free and open source software, thus one might want to host it on a different server). This part is optional if you use the official Neocities server.

For example, if your Website name is *example* and you wish to publish it on the official Neocities server, you can just set the `html_remote` to `neocities:example` or `neocities:example@neocities.org` if you want to be precise (but again, `neocities.org` is the default value for the `server_name`, so you can skip it).

To use the Neocities API, you also need to store your password. As you may want to track your fronde config file in git or other system, this secret value is not stored in the main config file. Instead, you have to put it in a specific `.credentials` file in the same folder as the config file. This `.credentials` file is a YAML file too. The credentials file is expected to contains one key per Neocities website, named `<website_name>_neocities_pass`. For exemple, if your Neocities website name is *example*, you must have a key named `example_neocities_pass` in this file.

```
---
example_neocities_pass: password
```

Listing 17: Content of an example `.credentials` file

3.1.5 `lang`

The main locale your website will be written in. Its value must comply with the ISO 639-1 standard.

```
---
lang: en
```

Listing 18: Example of a lang setting

The default `lang` setting is that of your current computer.

3.1.6 `sources`

The `sources` option stores an array of all source folders, where your org files to be published are. This enables you to gather from various different places (even at some absolute path in your computer) in your website.

Each source listed in that array is an object, which must use the following keys:

path [string] path to the folder containing the file to exports

name [string, optional] key used to generate the Org "project" name. This name never appears in a generated file, it is more like an internal id. Defaults to the last dirname of the path value.

title TODO

recursive [boolean, optional] whether the path should be exported recursively. Defaults to True

exclude [string, optional] Regexp of files to not export for this source. Default to nothing (no files to exclude).

target [string, optional] Path where to put the exported files, relative to the `html_public_folder` or `gemini_public_folder` folder. Defaults to the source name at the root of the *public* folder.

is_blog [boolean, optional] Whether this source should be considered as a blog and thus, serves to generate blog index, tags indexes and atom feeds.

theme [string, optional] Theme name to use for this source. Defaults to "default".

type [string, optional] Type of the exporter to use. Fronde currently supports only "html" and "gemini". Defaults to "html".

> If one of your sources is expected to use all default settings, instead of an object, you can just provide the source path as a string. Look at the `src` source in the example bellow.

```
---
sources:
- src
- path: src/news
  is_blog: true
```

Listing 19: Example of a sources array setting

If you don't provide any `sources` option, it fallbacks to support only one source folder named `src` at the root of your project directory, as if the `sources` option has been:

```yaml
---
sources:
- path: src
  target: .
  name: src
  recursive: yes
```

Listing 20: Default value of the sources option

Not setting the `sources` option is different from giving it the following exact value:

```yaml
---
sources:
- src
```

In the first case, as nothing as been given for the `sources` option, its default value will use `.` as its target directory, which means to put generated HTML files directly at the root of the `html_public_folder` (which defaults to `public_html`). But in the later case, the generated target will be the `src` folder *inside* the `html_public_folder`, which will be, by default, `public_html/src`.
Thus, if you have a very simple website with only one source, you should avoid setting the `sources` options, or be very precise in what you specify.

3.1.7 `templates`

The `templates` option enables you to customize the built HTML files, whatever source they come from. The main idea behind this is to add HTML fragments to the generated files.

The `templates` option stores an array of *template*, each one documenting an HTML fragment to insert or move at some place in a specified generated file. Each template listed in that array is an object, which must use the following keys:

type [string] how the template should be inserted or moved in the HTML document. Can be either `before`, `after` or `replace`. It defaults to `after`.

content [string] the HTML fragment to insert or replace. This value will be evaluated before insertion and some tags will be replaced in a context sensitive

manner. See the Preamble/Postamble/Templates formatting tags section for the details.

selector [string] a CSS selector specifying where to insert the new fragment. For example, if this value is `#main p:first-child` and the current template `type` is `after`, then the current `content` will be placed *after* the first HTML tag `p` inside a container, which has an `id` of `main`.

path [string, optional] the current template will apply only to generated HTML files matching this glob pattern. This pattern must match against a published path. That means, for a HTML document stored at `public_html/some/folder/doc.html`, the path could be `/some/folder/doc.html` or `/some/folder/*.html`, but not `public_html/...`.

source [string, optional] The aim of this option is to target, with a CSS selector, any part of the HTML document, in order to move it elsewhere. See example below.

The `content` and the `source` options are incompatible: either you have a `content` or a `source`, never both of them. This is because either you want to *add* a new content to your document (using `content`) or you want to *move* an existing content of your document (using `source`). In both case, the destination is taken from the `selector` option.

For example, if you want to hide social media `meta` tags in the *head* of your generated HTML files, you can use the following `templates` setting, which will add a bunch of `meta` tags *before* the `title` of any generated HTML file:

Another example: if you want, for example, to move the generated Org table of content before the main `#content` div (by default, the Org publish process puts it *inside* this div). As we specified a `path` option, this replacement occurs only for HTML documents under the `/docs` folder.

3.2 Preamble, Postamble, and Templates formatting tags

This section documents the percent-tags you can use in the preamble or postamble of your sources, or in any of your templates. This tags will be replaced by their corresponding content when you will build your website. The values are context sensitive, taken from the currently evaluated Org file, or by default from the config of your project.

%a the raw author name.

```yaml
---
templates:
- type: before
  selector: title
  content: |
    <link rel="schema.dc" href="http://purl.org/dc/elements/1.1/">
    <meta property="dc.publisher" content="%a">
    <meta property="dc.type" content="text">
    <meta property="dc.format" content="text/html">
    <meta property="dc.title" lang="%l" content="%t">
    <meta property="dc.description" lang="%l" content="%x">
    <meta property="dc.language" content="%l">
    <meta property="dc.date" content="%I">
    <meta property="dc.rights" content="%L">
    <meta name="twitter:card" content="summary">
    <meta name="twitter:creator" content="@fsfe">
    <meta property="og:type" content="article">
    <meta property="og:title" content="%t">
    <meta property="og:article:published_time" content="%I">
    <meta property="og:url" content="%u">
    <meta property="og:locale" content="%l">
    <meta property="og:description" content="%x">
    <meta property="og:site_name" content="My wonderfull website">
```

Listing 21: Templates setting to add social media meta tags

```yaml
- type: before
  selector: div#content
  source: nav#table-of-contents
  path: "/docs/*.html"
```

Listing 22: Templates setting to move the table of content outside the #content div

%A the HTML rendering of the author name, equivalent to `<span class="author">%a</span>`.

%d the short date HTML representation, equivalent to `<time datetime="%I">%i</time>`.

%D the full date and time HTML representation.

%F the `link` HTML tag for the main Atom feed of the current file source.

%h the declared host/domain name.

%i the raw short date and time.

%I the raw ISO 8601 date and time.

%k the current Org file keywords separated by commas.

%K the HTML list rendering of the keywords.

%l the lang of the current Org file.

%L the license information, taken from the config file.

%n the fronde name and version.

%N the fronde name and version with a link to the project home on the name.

%o the theme name (`o` as in Outfit) of the current file source.

%s the subtitle of the current Org file (from `#+subtitle:`).

%t the title of the current Org file (from `#+title:`).

%u the URL to the related published HTML document.

%x the raw description (`x` as in eXcerpt) of the current Org file (from `#+description:`).

%X the description, enclosed in an HTML `p` tag, equivalent to `<p>%x</p>`.

3.3 Rake tasks

Like a lot of ruby project, fronde exposes some Rake tasks. You can discover them with the `rake -T` command directly from your website directory.

Those rake task offer fine grained control over some operation, like featuring a “dry-run” mode for the `sync:push` task used by the `publish` command.

Here is a list of the most interesting one:

3.3.1 `sync:pull`

This command do the opposite of the publication: it will fetch all files from your remote site. This can be usefull if you edit your website from different computers and want to avoid losing some manually edited stuff.

As fronde normally take care of everything, this should not be used very often, but in case you need it, now you know it exists.

3.3.2 `tags:name` and `tags:weight`

Those two commands display the list of the keywords you already used on your blog project. The `tags:name` command sort them alphabetically and the `tags:weight` one sort them by frequency. In both case the order is reversed to have the "beginning" (i.e. letter A or most frequent use) directy near the command line prompt.

3.3.3 `cli:zsh_complete` and `cli:fish_complete`

Those two commands generates autocompletion files to be used by your shell of choice.

ZSH The idea is to store the new autocompletion file in a path already known by ZSH, i.e. somewhere defined in its `fpath` variable.

You can add such a path like this for example (if you don't already do it):

```
$ mkdir -p ~/.config/zsh
$ echo '[ -d $HOME/.config/zsh ] && fpath=($HOME/.config/zsh $fpath)' >> .zshrc
```

Finally, just create the autocompletion file in that directory (or any other you choose):

```
$ rake cli:zsh_complete > ~/.config/zsh/_fronde
```

Fish The idea is to store the new autocompletion file in a path already known by fish. It automatically recognizes `~/.config/fish/completions` for that matter.

Thus if you didn't created it yet, you can do the following:

```
> mkdir -p ~/.config/fish/completions
```

Finally, just create the autocompletion file in that directory (or any other you prefer):

```
> rake cli:fish_complete > ~/.config/fish/completions/fronde.fish
```

3.3.4 `site:clean`

This task will identifies autogenerated files, which can now be deleted. For each of them it will ask you whether you want to remove it or not (by default it will do nothing).

Currently it lists old keyword files for your blog projects (when you stop using a given keyword, its related generated index is not automatically removed) and published html files which do not seem to have an Org source file in any of your projects (a legitimate case would be if you directly add yourself such an html file in the `public_html` folder).