Clerq Video Guide

generated by Clerq on November 18, 2019 at 13:53

using ‘pandoc.md.erb’ template

Contents

Foreword

In this short video course, I want to introduce you to my new Ruby gem called `Clerq`. It is the third implementation of the idea to hold user and software requirements in separated markdown files. Started from the requirements management perspective it becomes rather a general tool for managing thick documents. And it seems to a suitable solution to the problem for many people who do some kind of technical writing.

It worth be mentioned that this whole document also is developing with Clerq. You can find its sources at github.com/nvoynov/clerq-video-guide and get the latest version of the document at bin/Clerq Video Guide.pdf.

Purpose

The purpose of this document is to provide scripts for the "Clerq User Guide (Youtube)[]". The document provides additional information for the set of video lessons, provides links, and summarizes the knowledge gained.

Recommended reading for all people who write thick structured documents especially in the area of software development.

References

The main subject of this document is using the Clerq ruby gem.

There are few other software products are using in the video lessons:

- Ruby;
- Thor;
- Atom;
- Git
- Pandoc.

Overview

The following sections of the document contain four lessons that introduce one with different aspects of using the Clerq.

The first chapter CLI and Nodes Hierarchy introduces into using basic command-line interface and managing nodes hierarchy through nodes metadata.

The second Metadata and Templates provides information on how to prepare a project for writing big structured documents.

The followed chapter Scripting and Automation shows you how to develop quality rules for your projects and how to publish documents in different formats through `Pandoc`.

The last chapter Advanced scripting provides a few examples of developing quality rules for two or more related projects where the second project depends on the first one.

Part I

CLI and Nodes Hierarchy

Foreword

This chapter introduces one with basic using of Clerq CLI and building nodes hierarchy based on nodes metadata. You must have installed Ruby and the Clerq Ruby gem to do the exercise.

```
$ gem install clerq
$ clerq -v
```

Script

1 Create a project

Let start from scratch and create a new project called "Clerq in Action" and open your preferable text editor in the created folder. I'm using Atom

```
$ clerq new clerq-in-action
$ cd clerq-in-action
$ atom .
```

We will not stop on projects structure that described in README.md and just start creating content right away. Let's go to the `src` folder and create `foreword.md`, `afterword.md`, and `part I.md`; provide a header for each file and print some lorem.

2 Add hierarchy

When we have some content in our repository (`src` folder is intended for content) we can check our hierarchy with `toc` command. And to do that - type `clerq toc` in your console:

```
$ clerq toc
% Clerq in Action
[01] Afterword
[02] Clerq in Action
[04] Foreword
[04] Part I
```

What a mess! Let's fix it by establishing the right hierarchy by adding hierarchy metadata.

1. Write id in `clerq in action.md` in its header: `# [r] Clerq in Action`.
2. Provide `parent` attribute by adding `{{parent: r}}` followed its header for all remained files:

- `foreword.md`,
- `afterword.md`,
- and `part1.md`.

Let's check the hierarchy again

```
$ clerq toc
% Clerq in action
[01] Afterword
[02] Foreword
[03] Part I
```

We can see there that 1) all the parts of the document belong to the root node `Clerq in Action`; and all the parts were provided with auto-generated ids.

To provide the right order we need do the following staff:

1. Setup custom id for each of the parts:
   - `# [f] Foreword` in `foreword.md`;
   - `# [a] Afterword` in `afterword.md`;
   - `# [p] Part I` in `part1.md`.
2. Specify the right order in `clerq-in-action` by providing `order_index` attribute:
   - `{{order_index f a p}}`

Check the hierarchy again and bingo! Now our hierarchy just perfect!

```
$ clerq toc
% Clerq in action
[f] Foreword
[p] Part I
[a] Afterword
```

3 Build project

And there is the right time to build our project

```
$ clerq build
> 'bin/clerq-in-action.md' created!
```

Let's go and see the result…

And at this point, one of the good ideas is to place all our work in Git

```
$ git init
$ git add .
$ git commit -m "Initial commit"
```

Afterword

In this first lesson, we introduced the basic Clerq command-line interface and the rules on how to build a single hierarchy from many separate files.

It worth be mentioned that to get information about Clerq CLI you can just type `clerq help` and see the output and farther instructions on how to get help for a certain command.

Metadata and Templates

Automation by Scripting

Part II

Advanced scripting

Advanced examples

Afterword