vsuite Documentation

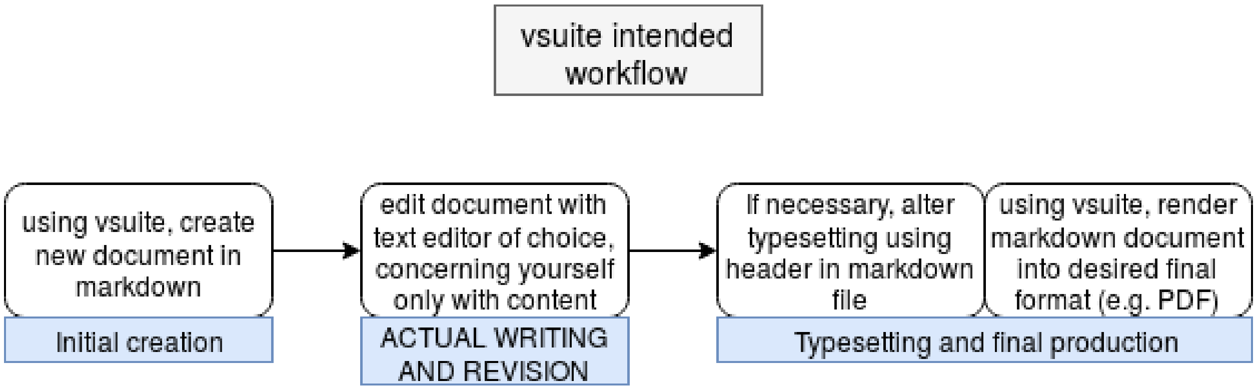
Release v0.4.0

Jesse Bulson-Lewis

Mar 06, 2018

Contents:

vsuite is a project management suite for Linux OSes aimed at writers who want tools that help–rather than hinder–them in their writing. At its core, it is a wrapper around various technologies that makes it easy to do your writing in markdown files that can easily be turned into finished final documents. Writing in markdown allows the writer, once she is up-to-speed on its usage, to separate the conceptually-distinct tasks of writing and typesetting in a manner largely inspired by this essay. Take a look at the following workflow diagram, this demo, and the *quick start guide* to see if you might benefit from using this software.


vsuite intended workflow
using vsuite, create new document in markdown
Initial creation
edit document with text editor of choice, concerning yourself only with content
ACTUAL WRITING AND REVISION
If necessary, alter typesetting using header in markdown file
using vsuite, render markdown document into desired final format (e.g. PDF)
Typesetting and final production

Note: vsuite is considered to be in early alpha, and as such should not be considered reliable yet. With that said, since it is a wrapper and never puts itself in charge of deleting or overwriting any user data, the risk of using it should be very minimal. Furthermore, it should go without saying that you should always backup any data that matters.

CHAPTER 1

Why vsuite (and why not a word processor)?

1.1 The case for content versus presentation

If you're asking this question, you probably use a WYSIWYG editor like Microsoft Word or LibreOffice Writer for most of your writing needs. While there are certainly use-cases where such programs are good tools for the job, there are also plenty of use-cases where they aren't. This essay provides a great analysis of the failings of word processors, and this core point is largely the driving force behind the creation of vsuite:

> Preparing printable text using a word processor effectively forces you to conflate two tasks that are conceptually distinct and that, to ensure that people's time is used most effectively and that the final communication is most effective, ought also to be kept practically distinct. The two tasks are
>
> 1. The composition of the text itself…
> 2. The typesetting of the document…the way in which structural elements will be visually represented…
>
> The author of a text should, at least in the first instance, concentrate entirely on the first of these sets of tasks. That is the author's business…
>
> I am suggesting, therefore, that should be two distinct "moments" in the production of a printed text using a computer. First one types one's text and gets its logical structure right, indicating this structure in the text via simple annotations…Then one "hands over" one's text to a typesetting program…
>
> —Allin Cottrell - Word Processors: Stupid and Inefficient

The message is that **authors of text should focus on content instead of the text's presentation**. Focusing in this way allows one to focus on one thing at a time and more effectively use one's own mental resources. Essentially, allow yourself as an author to not become distracted by presentation at the expense of content, which includes the text itself along with its logical structure.

Word processors are not designed around the paradigm of separating content from presentation. If you find yourself able to achieve separation of the two in a word processor by tweaking your existing workflow, more power to you. It seems likely, however, that you will still spend lots of time wrestling with your word processor to get it to do what you want (although you're probably used to such fights at this point).

1.2 Focusing on content

If we writers were to focus on the text and its raw words themselves, our writing would less clearly expose its own logical structure and then the job of the author would be only partially completed. However, we can indicate logical structure with only the slightest bit of effort using a "markup language" which is used to mark up raw text with symbols that denote logical elements (e.g. section headings). Markdown is an especially simple markup language that should serve many writers well.

We now need to know how we will concretely:

1. Author markdown text

   This will be done with a text editor of your choice.

2. Turn that markdown text into a presentable piece (e.g. a nice-looking PDF)

   This will be done with Pandoc, a tool that translates various text formats into each other.

However, using a text editor to create markdown documents (including their headers) from scratch and running Pandoc manually to create final documents has two problems: the new tools might be overwhelming to those just leaving the world of word processors, and there is a lot of repetition in creating new documents that can be eliminated.

1.3 Using vsuite to simplify writing in markdown

vsuite aims to generally ease the process of moving from nothing to a finished document, exposing the functionality of the underlying tools through a unified interface. It simplifies the initial creation of markdown files and finished documents, and also provides a structure for keeping track of bibliographical information.

CHAPTER 2

Installation

For the time being, vsuite is only distributed through its git repositories, and so should be installed with pip after installing a few software dependencies.

2.1 Required software

- `pandoc` (for rendering markdown to other formats)
- `pandoc-citeproc` (for citations in markdown)
- `git` (for optional versioning)
- `make` (for simplifying rendering of markdown)
- `pip3` (for installing vsuite)

2.1.1 Ubuntu 16.04

```
sudo apt install pandoc pandoc-citeproc git make python3-pip
```

2.1.2 Fedora 27

```
sudo dnf install pandoc pandoc-citeproc git make python3-pip
```

2.2 vsuite

Install vsuite as a Python package using pip3 pointed at its git repository:

```
pip3 install --user git+<URL of this repo>
# For example, using the GitHub repo
pip3 install --user git+https://github.com/jessebl/vsuite
```

The program will place files in `~/.local/share/vsuite` and store its config in `~/.config/vsuite`. The requisite files should be placed when the program is run, but this has not yet been tested to any rigorous degree. Pip also creates a vsuite executable named `vs` at `~/.local/bin/vs`, but if that location is not not a part of your PATH, you will need to manually add it.

CHAPTER 3

Quick Start

You might want to check out this demo for an example of the steps in this quick start.

3.1 Initialize a project with `vs init`

vsuite uses the concept of a project directory, in which you have various vsuite docs (which are simple pandoc markdown files) accompanied by a hidden `.vsuite` directory that holds accompanying files like CSL files and document templates. To get started using it, you need to initialize a directory as a vsuite directory:

```
vs init
```

This effectively creates an empty bibliography file, initializes a git repository, and creates the `.vsuite` directory which includes a project config file.

3.2 Create a new markdown document with `vs new`

Finally, you can get started with actually creating markdown files using vsuite:

```
vs new <document title>
```

This will create a file `<document title>.md`, after dropping or modifying spaces or some special characters (since GNU make really struggles with these things…). This file is generated from a template by vsuite and includes a YAML header that specifies fields for pandoc. This file is the one that you are meant to edit and do your work in. Tuning your text editor for use with markdown will be greatly helpful in this, since the whole point of this writing paradigm is to leave you, the writer, with more time doing actual writing. (For example, see this vim configuration file.)

3.3 Render your document with `vs make`

When you're ready to turn your markdown source into files for use by others:

```
vs make <project name>.<file extension of desired format>

# E.g generate a PDF of your document file "best_document.md"

vs make best_document.pdf
```

This uses GNU make along with a makefile in `.vsuite` to freshly generate the specified file unless it has been updated more recently than the source markdown file. Hence, you can always make sure that you have up-to-date documentation with `vs make`. The currently available formats are:

- pdf
- odt
- docx

CHAPTER 4

Code Documentation

4.1 User module

class vsuite.user.**User**

Represent a single user's vsuite installation

Track and manage vsuite's data files, user config, and more

get_fullname()

Get user's full name from /etc/passwd

Returns user's full name

Return type str

get_user_config()

Get user's user vsuite config

Get existing config if it exists Get and save newly-generated config if it doesn't

Returns user's user configuration

Return type configparser.ConfigParser

init_project_skel()

Create user data from vsuite skeleton

init_user_config()

Initialize user config

Create new config, refusing to overwrite existing one

Returns user's user configuration

Return type configparser.ConfigParser

read_user_config()

Get existing user config

user_init()

Ensure existence of user config and user data

4.2 Project module

class vsuite.project.**Project**(*path=False*)

Represent a project directory

The present working directory is considered to be the vsuite project root, unless one of its parent directories is a project, in which case that directory is considered to be.

Initializing sets attributes about where various directories and project resources would be, if they exist.

create_doc(*title*, *template_opt=None*)

Create new document with title name from template

Parameters

- **title** (*str*) – Title of document, used as basis for filename
- **template_opt** (*str*) – Document template to override default template set in project config

Returns Document filename

Return type str

Raises (FileExistsError) – If file with same name already exists

get_project_dir(*cursor_dir='/home/docs/checkouts/readthedocs.org/user_builds/vsuite/checkouts/latest/docs/source'*, *path=False*)

Absolute path to consider as project directory

Use present working directory if no parent directory is a project directory.

Parameters **cursor_dir** (*str*) – directory to check for project

Returns absolute path

Return type str

get_relpaths()

Get paths of project resources relative to pwd

Note: Mostly legacy method, preserved only using all assets' relative paths in `self.create_doc()`

Returns relative paths of project paths (e.g. the project's csl_dir)

Return type dict

get_template(*config*, *template_opt*)

Get template object to use for new document

Parameters **config** (*ConfigParser*) – config of project

Returns template to use

Return type jinja2.environment.Template

git_init()

Initialize git repository in project_path

init()
: Initialize project directory

  Reinitialize vsuite for the user, and initialize the present working directory as a project directory. This includes creating the .vsuite directory, creating and empty bibliography file, and initializing a git repo.

init_inherit()
: Initialize project directory

  Reinitialize vsuite for the user, and initialize the present working directory as a project directory, inheriting assets and config from the parents project

make(*output*)
: Use make and pandoc to generate outputs

  Use makefile from .vsuite directory, which leverages pandoc to generate requested outputs

  Parameters **output** (`str`) – name of argument to pass to make

4.3 Asset module

In the process of transitioning to the concept of an "asset" for use within the project module.

class `vsuite.asset.`**Asset**(*name*, *relpath*, *file_expression*, *project_path*, *data_dir='.vsuite'*)
: Represent a category of vsuite assets

  Parameters

  - **name** (`str`) – name of the asset (e.g. "bibliographies")
  - **relpath** (`str`) – path to assset directory relative to *project_path*
  - **file_expression** (`str`) – expression to match asset files, parsed by glob.glob
  - **project_path** (`str`) – path to project or other reference directory
  - **data_dir** (`str`) – directory within `project_path` to hold data files (`.vsuite`, leave unless you know what you're doing)

abspath()
: Get absolute path to asset

  Parameters **project_path** (`str`) – absolute path to project

  Returns absolute path to asset

  Return type str

abspaths()
: Paths of available assets

  Get absolute paths of asset files in first level of asset directory

  Returns file paths

  Return type tuple

copy_to(*dest_asset*)
: Copy asset files from self to another asset

  Parameters **dest_asset** (`vsuite.asset.Asset`) – asset to receive files

files()
: Available asset files

  Get asset filenames in first level of asset directory

  Returns asset filenames

  Return type tuple

print_files()
: Print asset files, newline delimitedfile names

relpath_pwd()
: Get path to asset relative to current directory

  Parameters **project_dir** (*str*) – absolute path to project directory

  Returns relative path to asset

  Return type str

Python Module Index

Index