Draft

Release 0.1.0

Jan 03, 2020

Contents

Draft is a CLI-enabled *writing system* to keep your work modular, optimize your project for git for version control, and just make writing more *fun*.

CHAPTER 1

Primary Features

When you're writing something short, a Google Doc or Word file works great. But get beyond a few thousand words and you end up in scroll-hell (i.e., the 'thumb' of the scrollbar is tiny and just navigating the doc is a chore).

Draft turns that half-finished novel into a clean, plaintext file tree perfect for git.

- `parse` any Markdown file into a file tree of Sections, Chapters, Sub-chapters, and Scenes
- `sequence` your files and directories to maintain a sequential index
- `outline` your project to help plan and fill in gaps
- `compile` your Scenes, Sub-Chapters, Chapters, and Sections into a single document once you're ready to publish
- Other features include `stats` (word count, etc.), `trim` (remove duplicate spaces), and `split` (put each sentence on its own line)

CHAPTER 2

Installation

```
pip install draft-cli
```

CHAPTER 3

Documentation

https://draft-cli.readthedocs.io/en/latest/

Source code:

CHAPTER 4

Contents

4.1 Draft Writing System

When you're writing something short, a Google Doc or Word file works great. But get beyond a few thousand words and all of a sudden opening that file comes with a sense of dread.

4.1.1 Philosophy

1) Writing in *modules* (or 'scenes' in Draft lingo) is easier and more fun than working out of a single large document.
2) Writing in plaintext is a more pure and distraction-free experience than WYSIWYG word processors.
3) Git is too useful a tool to not use in a writing project.

4.1.2 Organization

Folders

A Draft project is made up of a simple file tree as shown below:

```
whalebook/
└─project/                                  Organization:
│  └──Moby Dick Or The Whale/               [Title]
│     └──01-Nantucket/                      [Section 1]
│     │  └──01-Meeting Ishmael/             [Chapt 1]
│     │  │  └──01-Loomings/                 [Sub-Chapt 1]
│     │  │  │     01-His Name is Ishmael.md [Scene 1]
│     │  │  │     02-Habit of Going To Sea.md [Scene 2]
│     │  │  └──02-The Carpet-Bag/          [Sub-Chapter 2]
│     │  │  │     03-Old Manhatto.md       [Scene 3]
│     │  │  │     etc.
```

(continues on next page)

(continued from previous page)

```
│       │   └──03-etc.
│       └──02-Shoving Off/                    [Chapter 2]
│           etc.
└──02-Whaling/                                [Section 2]
    etc.
└──settings.yml
└──.gitignore
```

You don't *need* to use every 'level' of the project – i.e., you could just have the Title and a bunch of scenes, only use sections and scenes, have some scenes in sub-chapters and some scenes in chapters, etc.

The only required elements are the `project/ folder, the title/ folder (here called Moby Dick Or The Whale/), and scene `.md files.

Sequencing

Prepend 01-, 02-, etc. to your folders and files to keep them cleanly sequenced.

As you get to have a lot of folders and files, *re-sequencing* can get to be a pain (e.g., if you have 50 scenes and decide to split scene 02- into two separate scenes, you'll need to *re-sequence* the original 02- to be 03- and so on all the way to 51-).

Fortunately, just run draft sequence and Draft will auto-resequence for you.

4.1.3 Compiling

Each directory level corresponds to a Markdown heading level. When the project compiles, the indices (e.g., 01-) are stripped out and each folder's title is inserted as a heading (note: *Scene* titles are ignored).

By running draft compile, the above folder structure would translate into a Moby Dick, Or The Whale.md file with the following contents:

Moby Dick, Or The Whale

Meeting Ishmael

Nantucket

Loomings

Call me Ishmael. Some years ago — never mind how long precisely — having little or no money in my purse, and nothing particular to interest me on shore, I thought I would sail about a little and see the watery part of the world. It is a way I have of driving off the spleen and regulating the circulation.

4.1.4 Writing

Writing with Draft works best under the following guidelines:

1. **Use Markdown.**
2. **One sentence on each line.**

   This isn't required – it won't break anything! – it's just better for tracking changes in Git
3. **Save hash-based Markdown headings for separating sections.**

   If you need to use big font for whatever reason, stick to other header conventions (e.g., ========)

4. **Use Git and Github, and Commit often.**

   Git is incredibly useful in a writing environment and it's branching feature is a godsend if you want to try something radical (e.g., what if we switched from first to third person?). Github is a great visualization tool and provides a Cloud storage option for your project. And COMMIT-ing often is just good hygiene.

5. **Use a text editor with soft-wrapping and Markdown preview.**

- **Soft-wrapping:** Keeps your single-line sentences from running off of the page
- **Markdown preview:** See how your text translates into Markdown
- Atom has both of these features

4.2 Getting Started

4.2.1 Prerequisites

- Python 3 (required)
- Git (not required, but recommended)
- Virtualenv (not required, but recommended)

4.2.2 Starting From Scratch

Creating a new project from scratch is easy. Let's assume you wanted to create a project titled *Catcher In the Rye*.

1. Create project folder: `mkdir catcher`
2. Create virtualenv: `virtualenv rye`
3. Activate virtualenv: `source rye/bin/activate`
4. Install Draft: `pip install draft-cli`
5. Create project: `draft create-project 'Catcher In The Rye'`

You will now have the following file tree (note that Draft truncates your root directory to `catcher-rye/`):

```
catcher-rye/
└─project/
│  └─Catcher In The Rye/
└─settings.yml
└─.gitignore
```

4.2.3 From a WIP Project

1. Follow the same steps as "Starting From Scratch."
2. Take your current project in whatever format it is in, and paste it into a Markdown file and put in your project's root directory (e.g., if your current project is `myproject.md`, you should put it in `catcher-rye/`, in the same folder as `project/`).
3. Edit your file's Headings to follow Markdown conventions. This will inform your file tree:
   - # for the Title (e.g., # Catcher In The Rye)
   - ## for each Section

- ### for each Chapter
- #### for each Sub-chapter
- ##### for each Scene

For example, your `myproject.md` file might look like this:

> # Catcher In The Rye
>
> ## Meeting Holden
>
> ### Leaving School
>
> #### Roommates
>
> ##### We Meet Holden
>
> If you really want to hear about it, the first thing you'll probably want to know is where I was born, and what my lousy childhood was like, and how my parents were occupied and all before they had me, and all that David Copperfield kind of crap, but I don't feel like going into it, if you want to know the truth.

Note: You can mix and match headings as needed (or not use them at all) – but *scene* headers are needed if you want to split into multiple files

---

4. Run `draft parse myproject.md` which will take your file and split it into Section, Chapter, and Sub-Chapter *folders* and Scene Markdown files.

   E.g., the above `myproject.md` would result in:

```
catcher-rye/
└─project/
│ └─Catcher In The Rye/
│   └─1-Meeting Holden
│     └─1-Leaving School
│       └─1-Roommates
│         └─1-We Meet Holden.md
└─myproject.md
└─settings.yml
└─.gitignore
```

   If it doesn't turn out exactly as you want, no biggie! `myproject.md` is preserved, so just make whatever tweaks you need and re-run `draft parse myproject.md` to update the tree.

4.3 Commands

Using Draft's commands is easy. In your terminal just use `draft COMMAND`. If you ever need additional help while using the tool, just use `draft --help` or `draft COMMAND --help` to get information on specific commands.

Commands can be broken out into three buckets:

1. **Create** a project
2. **Format** a project's file content
3. **Maintain** a project

4.3.1 Create

Create commands are helpful when first starting up a project (`parse` especially for legacy projects).

`draft.cli.`**`create_project`**(*title*)

Generates a project structure. The root file name will be the title but: lower case, spaces replaced with dashes, and only the first two words (minus articles like 'and', 'of', etc.)

Parameters **`title`** (*str*) – Title for the project.

Returns None

Usage:

```
>>> draft generate-project 'Catcher In The Rye'
```

Example file tree:

```
catcher-rye/
└─project/
│ └─Catcher In The Rye/
└─settings.yml
└─.gitignore
```

`draft.cli.`**`parse`**(*filename*)

Generates a project tree based on a Markdown formatted file. Useful for generating project trees based on legacy projects or an outline file.

Will automatically strip punctuation out of Markdown headers for folder names, but will preserve them in the 'overrides' section of settings.yml for use in compiling.

Parameters **`filename`** (*str*) – Filename (i.e., not the full path) to be parsed (must include extension).

Returns None

Usage:

```
>>> draft parse mobydick.md
```

4.3.2 Format

Format commands are used to change the way text is arranged within files.

`draft.cli.`**`split`**(*filename=None*)

Splits multi-line sentences into separate lines. Affects all project files unless filepath is passed as an argument.

Parameters **`filename`** (*str*) – (optional) Filename (i.e., not the full path) to be parsed (must include extension).

Returns None

Usage:

```
>>> draft split '01-Meeting Ishmael.md'
```

`draft.cli.`**`trim`**(*filename=None*)

Removes all duplicate spaces from text. Acts on every file in project unless filepath argument passed.

Parameters **filepath** (*str*) – (optional) Filename (i.e., not the full path) to be parsed (must include extension).

Returns None

Usage:

```
>>> draft trim '01-Meeting Ishmael.md'
```

4.3.3 Maintain

Maintain commands are used mostly as you are writing, organizing, and publishing your project.

draft.cli.**sequence**()
Resets indices in folders and files and resolves duplicates.

Usage:

```
>>> draft sequence
```

Note: The number of digits in each file/folder *sequence* is governed by the total number of items at that *level*. For example, if you have 5 sections and 150 scenes, your sections will sequence 1, 2, 3, 4, 5 and your scenes will sequence 001, 002, 003, etc.

---

draft.cli.**stats**()
Gets statistics from the project (e.g., word count, etc.)

Returns word_count, scene_count, sub_chapter_count, chapter_count, section_count

Return type int

Usage:

```
>>> draft stats
>>> Words: 54034
>>> Scenes: 67
>>> Sub-Chapters: 30
>>> Chapters: 10
>>> Sections: 3
```

draft.cli.**compile**()
Compiles the project into markdown and HTML documents.

Usage:

```
>>> draft compile
```

draft.cli.**outline**()
Generates a new project outline.

Usage:

```
>>> draft outline
```

4.4 Settings

4.4.1 Overview

You can configure your project's settings with the `settings.yml` file in your project's root (note: this is automatically added for you with the `generate-project` command).

A few things to note before getting into the specifics:

> The `settings.yml` is not required – if you want to keep the default options, you can just delete it
>
> None of the *individual* settings in `settings.yml` are required. Omitting a setting will just use the default value

4.4.2 settings.yml

The default file layout (i.e., the one added automatically with `generate-project`) is below. Settings are split into four groups:

```yaml
headers:
  section: true
  chapter: true
  sub_chapter: true

warnings:
  parse: true
  split: true
  sequence: true
  trim: true

overrides:

author:
```

headers

headers defines whether or not each header type is displayed when using the `compile` feature.

This is useful for when you want to use the *organization* of the file structure without it affecting your *final product* (e.g., I might want to split my chapters up into sections to help keep track of them, but not present those sections to the reader).

warnings

warnings allows you to enable/disable the warning messages that come up during commands that affect the content of your files.

overrides

overrides is used to override the Section, Chapter, or Sub-Chapter name during the `compile` command.

For example, if we use the `parse` command, the chapter name *Ahab's Leg* would be translated into a folder *Ahabs Leg* – note the lack of an apostrophe. When using the `compile` command, we would want to add that apostrophe back in. To do so, the `settings.yml` might look like:

```
overrides:
  Ahabs Leg: Ahab's Leg
```

Note: When using the `parse` command, any header names that require cleaning before being created will automatically be logged in the `settings.yml` file if it exists (i.e., the above example would have been added automatically if detected during `parse`)

author

author is used to add an author name during the `compile` command.

4.5 Examples

The `examples/` directory contains Markdown files that demonstrate Draft commands.

4.5.1 Getting Started

1. Create a new project: `draft create-project 'New Project'`
2. Navigate to the project's root directory: `cd new-project`
3. Download the `examples/` folder into the `new-project/` directory: `svn checkout https://github.com/edelgm6/draft/trunk/examples`
4. Copy all of the files in `examples/` into your `new-project/` root directory

4.5.2 Trying the Commands

parse, stats, outline, and compile

parse

`parse.md` contains the first chapters of *Moby Dick* and can be used to build out a starter file tree.

1. Ensure your terminal is navigated to `new-project/` root directory and that `parse.md` is in the same folder
2. Run `draft parse parse.md`

You should now find a starting file tree based on the contents of `parse.md` in the `project/` folder.

stats

3. Run `draft stats`

You'll now have a printout of the word count, section count, chapter count, and sub-chapter count

outline

4. Run `draft outline`

This creates `outline.md` which is a Markdown file showing the basic outline of your project.

compile

5. Run `draft compile`

This compiles your entire project into a single Markdown file, *Moby Dick.md*.

split

`split.md` contains a paragraph from **Moby Dick** where **each paragraph takes up a single line.** To make writing and change tracking in Git easier, we want to use `draft split` to put each sentence on its own line.

1. Ensure your terminal is navigated to the same folder containing `split.md`
2. Run `draft split split.md`

`split.md` will now have each of its sentences on a discrete line.

If you've already tried the *parse, stats, outline, and compile* example, you can run `draft split` to act on every file across the entire `project/` folder.

sequence

`sequence/` contains a project with files that have a duplicate sequence: - `Call Me Ishmael.md` and `Manhattoes.md` both have the same sequence of `1` - There is no `2` sequence

To show how the `draft sequence` command fixes this:

1. Make sure your `project/` folder is completely empty
2. Move `sequence/` *into* the project folder
3. Run `draft sequence`

You will be prompted to choose *which of the 1 indexed files should go first*. Whichever you choose will become the new *1* sequence and the other will be come the *2* sequence.

4.6 About

This all starts with a half-finished novel that I had sitting on my Google Drive for the past eight or so years. Every once in a while I'd open it in some flash of inspiration or plain curiosity and go through the same two emotions each time:

1. **Excitement** at seeing some passages that struck me as particularly good at which point I'd start to think that it was worth finishing
2. **Dread** at the though of working through and reorganizing this monolithic document of 50,000+ words

There was clearly something of value there, but the thought of trying to make sense of it was overwhelming.

I knew I needed a software solution to help me break this up and do some radical surgery. I wanted something that fit all of the following criteria:

- **Modularity:** I should be able to break my project up into smaller pieces and *move them around* easily
- **Organization:** I should be able to organize my smaller pieces into a structure of sections, chapters, etc.
- **Internet connection not required:** I shouldn't need an internet connection to start writing. An online-first platform is distracting at best (checking your Twitter account is only a few clicks away) and disruptive at worst (anyone who's tried editing a Google doc on a spotty airplane wifi can attest to this)
- **Git/Github-friendly:** To make this into a coherent novel, I didn't just need version control, I needed *branches* and a way to store it in the Cloud

There are a lot of potential solutions out there, most notably Scrivener, which fits requirements 1-3 but not 4, and a bunch that fit 1 and 2 but not 3 or 4.

So I decided to roll my own which brings us to Draft.

Python Module Index

d

Index