Tom's Top 10 Asciidoctor Tips

by Tom Swan

I like simple things. Eating a grapefruit in the morning sun. Watching a spider build a web (how simple it seems for her). Coffee in peace and solitude.

Plain text editing.

I write in plain text. To create finished output (as for this article), I process my text files with Asciidoctor, a program that turns plain text documents into finished, publish-quality HTML and PDF files. Simple as that.

Over the past few years that I've used Asciidoctor, I collected some of my preferred practices into a list. Browse to the link below for my **Top 10 Asciidoctor Tips,** or read on for a brief Asciidoctor introduction.

Go directly to: My Top Ten Asciidoctor Tips

Asciidoctor in Brief

AsciiDoc markup is as simple as can be, and yet along with the capable Asciidoctor program, is seemingly unlimited in its ability to create beautiful, publish-quality PDF and HTML documents for viewing and printing and browsing. Just insert a few *markup* commands into the original source text, run the file through Asciidoctor, and your finished document is ready for publishing. This article is its own example — I wrote it entirely using AsciiDoc markup, as I do my entire web site.

With this marvelous tool, I have created what I call my dream writing system, *Web Writer*. A series of scripts and organizations make it possible for me to create entire web sites (such as tomswan.com), write articles, books, illustrate my music scores, keep notes and to-do lists, and do just about everything else that involves writing. I almost never use Word, Pages, or Open Office word processors anymore — I now write entirely in plain old text, a fact that alone has helped me to be more productive and get better results faster than ever before.

I love Asciidoctor. Can you tell?

> AsciiDoc is a simple but powerful scripting or *markup* language for plain-text writing. Asciidoctor is a program that converts AsciiDoc documents from text to HTML and PDF formats for viewing on- and off-line, for printing, for assembling into books, for writing blog postings such as this one and for creating many other types of documents, even entire web sites.

Is Asciidoctor right for you? Maybe, maybe not. It's ideal for techies and computer types like me, but you don't need to be a crack C++ programmer to use Asciidoctor.

Those who do, however, including me, realize several of the benefits of writing in plain text instead of having to master a word processor's proprietary formats and protocols:

- **Every computer everywhere** has a text editor. You can potentially create and edit your documents on any system that can access your files. Store your files in a cloud Dropbox account and you create an instant edit-anywhere, self-publishing system! Update your novel in a public library and continue back in your office seamlessly with no regard for whether the proper software is installed on the foreign system.
- **Plain ASCII text** is unquestionably the most-common data format in existence. Your documents remain read- and write-ready regardless of future advances in computer technology and software. ASCII, it's safe to promise, is here to stay.
- **Your original source text** is perfectly readable by itself — try that with a word processor's raw file data. Even the most ancient computer system can probably handle the article you just wrote — a fact that is rarely possible with proprietary word processor formats. (Try opening new word processor documents on that old laptop you have in the closet — nope, that probably won't work.)
- **Working in plain text** helps you to focus on what you are writing, rather that how the finished document will look. All visual elements such as header styles and tables are controllable with stylesheets, but the defaults will work great for most people. I mostly use default stylings with only a few minor adjustments. (The default stylesheets are also mobile ready — all elements adjust automatically for different display types. Resize your browser window to see this effect in action.)

Asciidoctor and its PDF-generating counterpart Asciidoctor-pdf are command-line programs — what may seem "old school" to many. Before the mouse and graphical windows, a terminal was the only way to issue commands. Now, a command-line "shell" prompt is becoming a rarity, but for the computer savvy, it remains one of the best ways to "talk" to a computer and make it operate the way you want. I use a combination of GUI and command-line tools.

To process a document written with AsciiDoc markup, go to a terminal prompt, shown here as a dollar sign `$`, and type commands such as this (assuming you have installed Asciidoctor):

```
$ asciidoctor mytext.txt
```

That's all that's needed to create mytext.html (for a web browser). To create a PDF document, feed the identical text to the companion program:

```
$ asciidoctor-pdf mytext.txt
```

Open the resulting mytext.pdf or mytext.html in any web browser. It's that simple. Learn more about Asciidoctor, Asciidoctor-pdf, and the AsciiDoc markup language by following the links in this article. Install the program and try it out — it's entirely free to obtain and use.

My Top Ten Asciidoctor Tips

Following are some of the ways that I use Asciidoctor and the AsciiDoc markup language. I put most of these ideas into practice in every document that I write. The tips are in no particular order — I'm not trying to outdo Letterman here; I just want to share some information that I find useful.

1: Separate Macros and Text

Separate documents that have more than a few macro definitions into .adoc and .txt files that are otherwise named the same — MyStory.adoc and MyStory.txt for example. Put your macros and other settings such as the title and author into the .adoc file (I call them header files). Put your prose in the .txt file. Here is a portion of one of my .adoc header files:

```
// NO BLANK LINES ALLOWED
:author: Tom Swan
:email: <tom@tomswan.com>
:filename: test.txt
:sectanchors:
// Stylesheet settings
:stylesheet: github.css
:csspath: /css
```

After this point, blank lines are permitted. Until then, remember: *No Blank Lines!* End the .adoc file by including the text of your document using the following statement preceded by a blank line, which is now okay to use because we are done defining macros:

```
include::{filename}.txt[]
```

Notice that the `filename` macro is defined earlier and used by encasing it in curly braces. The double square brackets `[]` designate `include::` as a macro that, in this case, replaces itself with the contents of the named file after the double colon. The result is a single document for Asciidoctor to process. You can store text and macros together in a single file if you want, but keeping definitions and text separately makes it easier to share and swap common macros such as external URL references and various style settings.

Upon the first blank line encountered, Asciidoctor switches some gears internally which can affect the ability to alter certain macro values. Details of this are in the online documentation, but it is simply all around easiest just not to insert any blank lines until after you define most of your macro settings at the top of a document.

2: Preface Header Levels with Equal Signs

There are two ways to specify header levels in AsciiDoc. You can underline text as in

```
Gong With the Wind
------------------
```

Or you can preface the title with equal signs. This the same as above:

```
== Gong With the Wind
```

But while the underlined text looks good in raw form, the second is preferable because it clearly shows the level number. With the under-text method, you must remember or look up which character goes with the header level you want — most inconvenient.

Switch to `===` prefaced headers if you are used to the other now less popular style of designating header levels. Eventually your eyes will adjust to this header-formatting style, which I have found makes keeping my headers in outline order a lot easier.

Just change the number of equal signs to the level you want (up to six, more than enough for even the deepest outline).

To create a level-three head, for example, type

```
=== Title Text
```

Be sure that your header markup equal signs are flush left in the document with no leading spaces.

3: Use "Cut Lines" to Organize Text

To help distinguish sections easily in the source text, I insert "cut lines" (my term) between parts. I program function keys to trigger macros in my text editor (Sublime Text 3) to insert separator lines as comments that Asciidoctor skips. This really keeps my source text files looking clean. Asciidoctor ignores "commented-out" lines beginning with two slashes:

```
// --------------
```

Double the double slashes to delimit a "block" or multi-line comment, easier when you have many lines of text to hide:

```
////
All of this text
and the comment slashes are
ignored and hidden in the output
////
```

Single-line text comments beginning with double slashes such as

```
// Fix spelling errors below!!!
```

help remind you of tasks, pay the light bill, whatever. Programmers use comments to document code and for dirty laundry such as to-do lists and private notes not for publication. (Asciidoctor's

comment style is also used in C++ and some other coding languages by the way, not that it matters). Comments never appear in the finished goods.

4: Convert Headers to Anchors for Sharing and Bookmarks

Always define the `:sectanchors:` macro — no arguments are needed, just the macro name — so that bookmarks can be set on section titles. This makes it easy also for viewers to share locations within a document rather than only referencing the entire text from the top, for example when sharing a link on FaceBook.

Hover your mouse cursor over any header in this article to see section anchors in action. Right click on the pilcrow symbol that appears for a menu of options.

5: Define Macros for Special Characters

Define macros for special symbols and then enter text such as `{ul}filename` which comes out `_filename` in the finished document. Here's my list of macros I commonly define, some of which use Unicode symbols:

```
:copyright: ©
:cpp: C++
:dot: .
:ellipsis: …
:hatch: #
:pilcrow: ¶
:star: *
:tilde: ~
:ul: _
```

6: Use Macros for External URL References

Create macros for external URL references rather than strew these throughout the text (which always looks messy to me). For example, in the .adoc header file define macros such as the following:

```
:website: http://www.tomswan.com[tomswan.com]
:youtube: https://www.youtube.com/user/TomSwanPlaysGuitar[YouTube]
```

You can now simply write the macro name in curly braces like this:

```
{youtube}
```

And in the resulting document readers see the link text:

YouTube

7: Add Sidebar Titles Using Dotted Headers

Use "dotted" headers instead of section headers for a different kind of header that might top a sidebar or some text you want to identify in some special way. Just start a blank line with a dot and immediately follow with your text, no spaces, like this:

```
.[red]This is Noteworthy!
```

which comes out like this:

This is Noteworthy!

Use this format to add titles to `NOTE:` and other admonition blocks to which you want to draw attention, such as this one.

8: Redirect HTML and PDF File Output

Normally Asciidoctor creates output HTML and PDF files in the same path as the source text. To send output files elsewhere, use the `-D` output option. For example, this command:

```
$ asciidoctor -D /temp myfile.adoc
```

compiles myfile.adoc into myfile.html in the /temp folder. Otherwise, output goes to the current path where myfile.adoc is located. I use the `-D` option to compile various notes and other texts to a Dropbox shared folder so that I can view my publish-quality files from any computer with internet access. Doing that via a repeating crontab entry on my development system keeps my notes, logs, and other text documents up to date by the minute, even when I'm away from the office. Very handy.

The extensions .adoc and .txt are probably the most common filename extensions used to identify text documents, but you can use other names if you want.

9: Customize How HTML Anchors are Referenced

Define `idprefix` as null (no arguments); otherwise anchor names (also called "section IDs") are prefaced by the `idseparator` value, which defaults to an underscore. I change that to a dash using these two definitions in the no-blank-lines section of my .adoc header files:

```
:idprefix:
:idseparator: -
```

Given those settings, the title **Tom's Top 10 Tips!** becomes the anchor name `tom-s-top-10-tips`. Use double angle brackets to create links to the anchor location:

```
Go back to: <<asciidoctor-in-brief>>
```

That anchor, `asciidoctor-in-brief`, is defined automatically for this article's earlier section so titled. I didn't have to create the anchor explicitly in any way. Try it out by following the actual link as created by the above text (hit your back button to return):

Go back to: Asciidoctor in Brief

I very much dislike underscores in filenames and identifiers, except for prefacing special symbols such as `__DEBUGGING` where a unique name is advantageous. Use dashes in place of underscores — they are easier to see and to type.

10: Inspect Output with Developer Tools

Use a good web browser inspector such as FireFox's Web Developer tools (in the Tools menu if installed) to inspect the HTML that Asciidoctor generates. If something doesn't look right in your document, inspect its HTML — or isolate the problem section in a test file to examine. Often the solution to a puzzling formatting problem is easier to spot that way than by endlessly staring at the source text, a fact that I learned the hard and bleary-eyed way.

11: *BONUS:* If admonition blocks such as `TIP:` don't seem to be recognized in your editor's AsciiDoc syntax highlighter, the cause is invariably an underscore or another special character, maybe in an earlier source code block. Copy and move a faux `TIP:` up and down to find where the formatting fails. (In Sublime, Shift-Ctrl-Up and -Dn.) The error is close by. To help prevent the problem, define macros for special characters (see tip #5). PS: The preceding hatch mark is somewhat ironic.

Show Me the Text

In a related article I provide links to source text files of sample web pages that you can view in a web browser while you also inspect the source text exactly as I wrote it. The demonstration also includes samples of how to control the look of published AsciiDoc documents using CSS stylesheets. Read all about it here:

Continue…

Or,

Download or try the demo online…

Top of Page