
CTRL-ZINE
ISSUE 25 - VOL 3

About ^Z

Ctrl-ZINE (^Z) is a Ctrl-c.club/Smol Web collaborative zine that celebrates tech and the Smol Web. Started in March 2023, Ctrl-ZINE publishes a monthly issue, where anyone can download a PDF version and a pre-folded PDF version for home printing. No digital format of the content is maintained on a Website whatsoever. Some of the topics within these issues range from Smol Web protocols and communities (ActivityPub, Tildeverse), Web-adjacent protocols (Gopher, Gemini), alternative forms of communication (HAM radio, LoRa, finger), snippets of code, artwork, and anything tech-related

Those who contribute to ^Z are passionate about what they share. They want what is best for Us, the citizens of the Web. With that, anyone with that same passion is welcome and encouraged to contribute to future issues. Further info can be found in the Editorial section of this issue. May the Smol Web live forever!

ZINEHEAD Press
Editors (for this issue): ~loghead (loghead@ctrl-c.club), past editors: ~singletona082 (singletona082@ctrl-c.club)
Submissions contact: loghead@ctrl-c.club / x2600 (IRC)
Comments/questions: loghead@ctrl-c.club / x2600 (IRC)

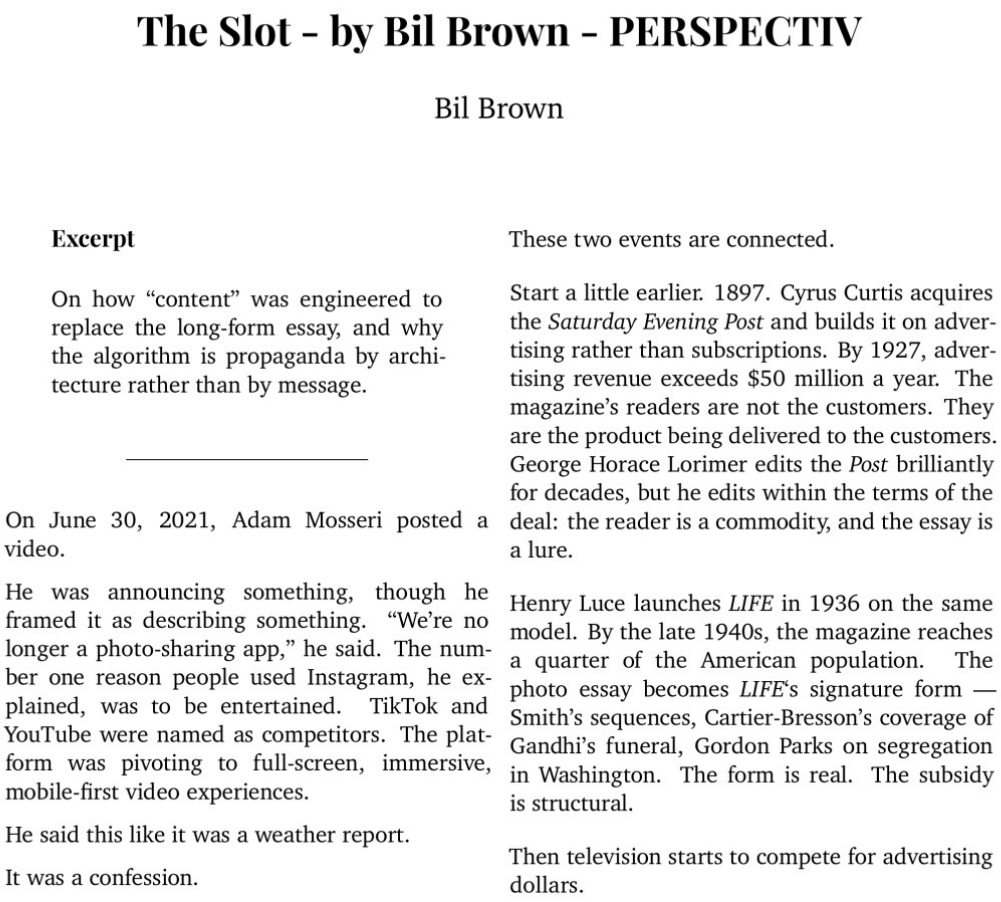
INDEX

From web page to printable PDF for reading later *by* baty

Forking The Web *by* ~nttp

I'm a blog, yeah! *by* ~loghead

From web page to printable PDF for reading later
by baty



The Slot - by Bil Brown - PERSPECTIV

Bil Brown

Excerpt

On how “content” was engineered to replace the long-form essay, and why the algorithm is propaganda by architecture rather than by message.

On June 30, 2021, Adam Mosseri posted a video.

He was announcing something, though he framed it as describing something. “We’re no longer a photo-sharing app,” he said. The number one reason people used Instagram, he explained, was to be entertained. TikTok and YouTube were named as competitors. The platform was pivoting to full-screen, immersive, mobile-first video experiences.

He said this like it was a weather report.

It was a confession.

These two events are connected.

Start a little earlier. 1897. Cyrus Curtis acquires the *Saturday Evening Post* and builds it on advertising rather than subscriptions. By 1927, advertising revenue exceeds $50 million a year. The magazine’s readers are not the customers. They are the product being delivered to the customers. George Horace Lorimer edits the *Post* brilliantly for decades, but he edits within the terms of the deal: the reader is a commodity, and the essay is a lure.

Henry Luce launches *LIFE* in 1936 on the same model. By the late 1940s, the magazine reaches a quarter of the American population. The photo essay becomes *LIFE*‘s signature form — Smith’s sequences, Cartier-Bresson’s coverage of Gandhi’s funeral, Gordon Parks on segregation in Washington. The form is real. The subsidy is structural.

Then television starts to compete for advertising dollars.

Sample PDF output

Rather than using a normal-person’s "Read Later" service, I print long-form web articles for reading later. I print them, pile them up, and read them all when I have some time away from the computer. It’s the only way I can truly pay attention to them.

The process took some work to dial in, but I’ve gotten it close to how I like it. It goes like this:

1. Save the page as Markdown using the Obsidian Web Clipper
2. Convert the Markdown to PDF using Pandoc
3. Print!

First, the Obsidian Web Clipper. I have a custom "template" configured for this. The template doesn't use Properties. I put the entire block of front matter into the Content field, using variables to insert the specifics, so no need to worry about the way the clipper renders properties…

```
---
title: "{{title}}"
author: "{{author|safe_name}}"
source: {{url}}
created: {{date}}
published: {{published|date:"YYYY-MM-DD"}}
---

> ## Excerpt
> {{description}}

{{content}}
```

A copy of the template for importing directly into the web clipper is here

The magic comes from Pandoc. I convert the Markdown to PDF via LaTeX. The default.latex template that ships with Pandoc works well. It uses all sorts of variables that can be passed in via YAML front matter or via a defaults file. I only just learned about the default file option, so that cleaned up the front matter dramatically. Here's my *~/.pandoc/defaults/article.yaml* defaults file:

```
pdf-engine: xelatex
template: ~/.pandoc/templates/default.latex


variables:
  documentclass: scrartcl
  mainfont: XCharter
  sansfont: Playfair Display
  linestretch: 1.15
  header-includes:
    - \setkomafont{title}{\sffamily\bfseries}
    - \setkomafont{section}{\sffamily\bfseries}
    - \setkomafont{subsection}{\sffamily\bfseries}
  classoption:
    - DIV=14
    - twocolumn
```

This sets fonts, sizes, document class, etc. The DIV=14 is new to me. With KOMA classes, it sets the text width, taking font size into account. This is easier than including specific geometry. Smaller numbers = narrower text. Drop the twocolumn line to print full-width.

I could create separate default files for different layouts and styles. Use the -d option, e.g. pandoc -d article. I might make one for use on the Remarkable tablet, which likes a larger font and wider body.

A shell script, md2pdf.sh does the conversion for me. The script used to be about 10 lines of sloppy bash with a bunch of stuff hard-coded. I had Claude help me make it more robust, and here's the latest version:

```
#!/bin/sh
set -eu


DEFAULT_DEFAULTS_FILE="$HOME/.pandoc/defaults/article.yaml"
```

```
usage() {
    cat >&2 <<EOF

Usage: $(basename "$0") [OPTIONS] FILE

Convert a Markdown file to PDF with pandoc and open it.

Options:
  -o DIR    Output directory (default: same directory as FILE)
  -d FILE   Pandoc defaults file (default: $DEFAULT_DEFAULTS_FILE)
  -n        Don't open the PDF after creating it
  -h        Show this help
EOF
    exit 1
}

output_dir=""
defaults_file="$DEFAULT_DEFAULTS_FILE"
open_after=1

while getopts "o:d:nh" opt; do
    case "$opt" in
        o) output_dir="$OPTARG" ;;
        d) defaults_file="$OPTARG" ;;
        n) open_after=0 ;;
        h) usage ;;
        *) usage ;;
    esac

done
shift $((OPTIND - 1))
```

```
[ $# -ge 1 ] || usage

input="$1"

if [ ! -f "$input" ]; then
    echo "Error: input file '$input' not found" >&2
    exit 1
fi

if [ ! -f "$defaults_file" ]; then
    echo "Error: defaults file '$defaults_file' not found" >&2
    exit 1
fi

# Default output dir to source dir
[ -n "$output_dir" ] || output_dir=$(dirname "$input")
mkdir -p "$output_dir"

stem=$(basename "$input")
stem="${stem%.*}"
output="$output_dir/$stem.pdf"

# Capture pandoc stderr so we can report it clearly on failure
log=$(mktemp)
trap 'rm -f "$log"' EXIT

if ! pandoc --defaults "$defaults_file" "$input" -o "$output" 2>"$log";
then

    echo "pandoc failed converting '$input'" >&2
```

```
    echo "----- pandoc output -----" >&2
    cat "$log" >&2

    echo "-------------------------" >&2
    echo "Command: pandoc --defaults '$defaults_file' '$input' -o
'$output'" >&2
    exit 1
fi

# Surface warnings even on a successful run
[ -s "$log" ] && cat "$log" >&2

echo "Created: $output"

if [ "$open_after" -eq 1 ]; then
    xdg-open "$output" >/dev/null 2>&1 &
fi
```

So the simple version (assuming md2pdf.sh is in your path) is:

```
md2pdf.sh my-article.md
```

It's not quite cross-platform (e.g xdg-open on Linux vs open on macOS) but that shouldn't be to difficult to do.

This seems like a lot, but once it's in place it's a couple of clicks and a quick shell command.

Let me know if there's anything unclear or incorrect here. Or if you have suggestions for improvements.

Reference links:

1. Obsidian (from *From web page to printable PDF for reading later*) https://obsidian.md/clipper

2. Pandoc (from *From web page to printable PDF for reading later*) https://pandoc.org/

3. Template *(from From web page to printable PDF for reading later)* https://static.baty.net/code/obsidia-web-clipper-for-pandoc.json

Forking The Web *by* ~nttp

A few days ago as of this writing, Rodrigo Arias Mallo, the maintainer of Dillo, published a blog post titled On forking the Web. That would be a good idea, because frankly? The "living standard", outside of being literally impossible to catch up with, means "whatever Google does in Chrome is now the standard". That's no way to govern the capital-w Web. But how to go about it?

Not being a browser implementer, my opinions are uninformed. Hopefully I'm not too far off the mark.

For one thing: we need to go back in time and start over. The HTML 4.01 spec was really nicely written, but already pretty big and complex. Let's go even farther back to HTML 3.2 and start over from there; it's defined in a single, stand-alone web page 125K in size (that I just downloaded to my hard drive).

Second: get rid of the false dichotomy between semantics and presentation; a heading doesn't cease to be a heading just because it has an align attribute. As for the center tag? I need it all the time in practice. Forcing me to use style=... instead is just performative and bureaucratic.

Speaking of styles: they're a great idea, but they should be purely client-side: i.e. something people install in their browsers and can apply to any site. Dark mode for example could be a special case of that instead of a specific browser feature.

How about scripts? I would hate to give up web games entirely for example. But imagine "applets" that are literally just Love2D in an iframe. Video and audio? Those never belonged in the browser anyway; let me click a thumbnail to open the link in my preferred media player instead. It's called separation of concerns.

HTML5 had some good ideas (though I'm not sure why we needed a <nav> element when dir and menu were right there). We don't have to throw the baby out with the bathwater. But we do need a modern web standard that stays put and can be reasonably implemented by multiple parties. The web is too important to not be a public good.

I’m a blog, yeah! *by* ~loghead

“All the young blogs (Hey, blogs!)
Carry the logs (Where are you?)
Dog of the blogs (Stand up)
Carry the logs (Ha-ha)”

(..the tune of Mott The Hoople “All The Young Dudes”)

You know, this is fitting. I’ve said a bunch of times (and likely will again), about how blogs and the blogging online ecosystem changed a lot over the years, and I jumped in at a damn odd time. Blogging, logs, journals, they overtook the Internet in the early-2000s, and then it became commercial (2004-ish). I started in 2006, ads right away. Google AdSense enabled this. Readership was low, but I got lucky and had big links within a week. Soon I had a small income from blogging.

Then commercial blogging died (social media). Then blogging, itself, damn near died (again, social media). And I would scour from link to link, daily, between blog posts, looking for something to add to RSS. I rarely found new material. I had maybe 15 (still) updated blogs on RSS, and maybe 5-6 entries on the entire feed for a week.

Fast forward: Small web. Hell, I couldn’t keep up with just a single day’s worth of entries from a single blogroll now. And all blogrolls unique. All loaded with amazing outlets, journals, logs.

Webrings, blog discovery tools, blog platforms — it’s like the Web/universe saw some deficiency in blogging and in some odd fashion caused the Internet to 180 back to the blogosphere. Almost as if the Web (anyone/everyone) saw and knew there were less of a thing that needed to be there, and was like: “ah, one quick shot will fix you all up!” Keyboard, text editor, Publish, blogosphere!

There’s more to it than that. Several years grew it to where it is now (and GROWING!).

Color me happy. I have to **refine** and **edit** an RSS feed now!