Hexdump (AWKzine Issue #2)

Todd Coram (todd@maplefish.com)

February 25, 2026

A Simple Hexdump utility in Gawk


```
BEGIN { VERSION="Hexdump Version 1.0" }
```

This file is a (semi) Literate Programming Document of Hexdump. [2] Hexdump is a program that will *dump* the contents of a binary file in hexadecimal. This file has no dependencies other than requiring `gawk` to run.

Hexdump is documented using Knit and AFT [3].

Contents

1 Front Matter: What do I do with this?

This file is meant to be read as well as executed. You may be reading the source file (hexdump.awk) or the the HTML or the PDF version.

To generate the HTML page (hexdump.html) of this file run this comand:

```
$ gawk -f knit.awk hexdump.awk >hexdump.aft && aft hexdump.aft && aft hexdump.aft
```

(Yes, yes... run `aft` twice, it needs a 2nd pass to build up a table of contents...)

The hexdump.awk file can be run like any normal gawk script. For example, to dump the hexadecimal values of this file itself you can do:

```
$ gawk -f hexdump.awk  hexdump.awk
```

You will see output that looks like:

[1] To view a copy of the license, visit [http://creativecommons.org/licenses/by-sa/4.0

[2] The single file source is likely to be found at http://www.maplefish.com/todd/hexdump.awk . This document can be found at http://www.maplefish.com/todd/hexdump.pdf and http://www.maplefish.com/todd/hexdump.html

[3] Knit is available at http://www.maplefish.com/todd/knit.html) and AFT is at http://www.maplefish.com/todd/aft.html. If you run Ubuntu then you may find yourself able to install AFT using apt install aft

```
00000000  23 20 2a 54 69 74 6c 65  3a 20 20 48 65 78 64 75
00000010  6d 70 20 20 28 41 57 4b  7a 69 6e 65 20 49 73 73
00000020  75 65 20 23 32 29 0a 23  20 2a 41 75 74 68 6f 72
00000030  3a 20 54 6f 64 64 20 43  6f 72 61 6d 20 28 74 6f
00000040  64 64 40 6d 61 70 6c 65  66 69 73 68 2e 63 6f 6d
00000050  29 0a 23 0a 23 09 09 41  20 53 69 6d 70 6c 65 20
00000060  48 65 78 64 75 6d 70 20  75 74 69 6c 69 74 79 20
00000070  69 6e 20 47 61 77 6b 0a  23 20 0a 23 09 09 20 7e
00000080  54 68 69 73 20 77 6f 72  6b 20 28 64 6f 63 75 6d
00000090  65 6e 74 20 61 6e 64 20  73 6f 75 72 63 65 20 63
```

(We truncated this to just the first 10 lines.) [4]

This is similar to `hexdump -C` on your Linux or BSD platform (but without the ASCII portion).

2 Why Do This?

What benefit does this have? Not much, except it's a proof of principle. I wanted to see if I could abuse gawk and coerce it into processing binary (byte oriented) data. It was a success and this technique will aid me in doing other nasty things that you shouldn't do in gawk.

But, again, why? Because gawk is fairly ubiquituous (or at least highly portable), small and has no dependencies or "batteries"' that need to be included. Awk (and by extension: gawk) is a nice concise language and we can do a lot of stuff with it. So... This is a short one, so let's just get to it!

3 Implementation

All of this is based on the feature in gawk that allows one to set the record separator `RS` to be a regular expression. In our case, we want the regular expression to be *any* single byte. In a regular expression a `.` represents a single character or byte.

```
BEGIN { RS="(.)" }
```

That's it. And we capture this character (that's what the parenthesis do) in a bulit in gawk variable `RT` so we can manipulate it. This is, by no means, efficient. But it's good enough for decoding small binary files (and eventually *payloads*) that will let us do some interesting stuff in future AWKzines.

Now we want to set up some variables to help us pretty print our output in the format we saw above. We use `cnt` to keep track of the number of bytes read so we can output the position (e.g. `00000010}). We keep track of whether or not we should start a` newline|. We also keep track of the number of columns (`col`) per line.

```
BEGIN { cnt=0; newline=1; col=1 }
```

Unfortunately, we can't just print out `RT`. It has a numeric value of 0 as it is the byte character read (as captured by `RS`). So, we have a table that maps every possible character/byte (0 - 255) as the numeric value 0-255. This maps the character to it's ordinal value.

To build this table (we call `ORD`) we have the following function:

```
function ord_init(    i, t) {
    for (i = 0; i <= 255; i++) { t = sprintf("%c", i);  ORD[t] = i }
}
```

We must call this function to build the global table `ORD`.

```
BEGIN { ord_init() }
```

Now we are ready to do the actual work (on the supplied file or `stdin`). First let's handle printing out the count/address or the line of bytes for every new line. We had set `newline = 1` earlier, so this is the first pattern matched.

[4] Yes, hexdump.awk is **not** a binary file, but this dump **is** a binary representation of it.

```
newline { printf("%08x ", cnt); col = 1; newline=0 }
```

Note that we (re)set the column `col` to 1 and clear `newline` until we are ready to match it again.

If you look at the output of the `hexdump` command, it prints out 8 bytes and then a blank column before printing out another 8 bytes. This is just visual sugar, and is easily accomplished with this pattern:

```
col == 9 { printf(" ") }
```

The actual hexadecimal value is printed out with a leading space. We print out the ordinal value of the character captured in `RT`:

```
{ printf(" %02x", ORD[RT]); col++ }
```

We want to print out 16 hexadecimal values before triggering a new line. So, if have printed 16 already (and incremented `col`) then we trigger `newline` for the next record (byte) read.

```
col == 17 { printf("\n"); newline = 1; cnt += 16; col=1 }
```

That's it. We could get fancy and print out the ASCII characters after the 16 hexadecimal values, but that is overkill and left as an excercise to the reader :)

```
END { printf("\n") }
```

4 End Matter

That's it. All of this was just so we can stretch our legs and gear up for some future AWKzine where we will do binary stuff (e.g. binary protocol decoding, etc) that really abuses poor old gawk.

5 AWKZine

AWKZine? Is this a Zine? Well, in my very loose interpretation: Yes. I want to show you how to do stuff... in Awk. It's fringe. It's fun.

This was Issue #2. Hope you enjoyed it!

6 Revisions

version 1.0 - Initial release.