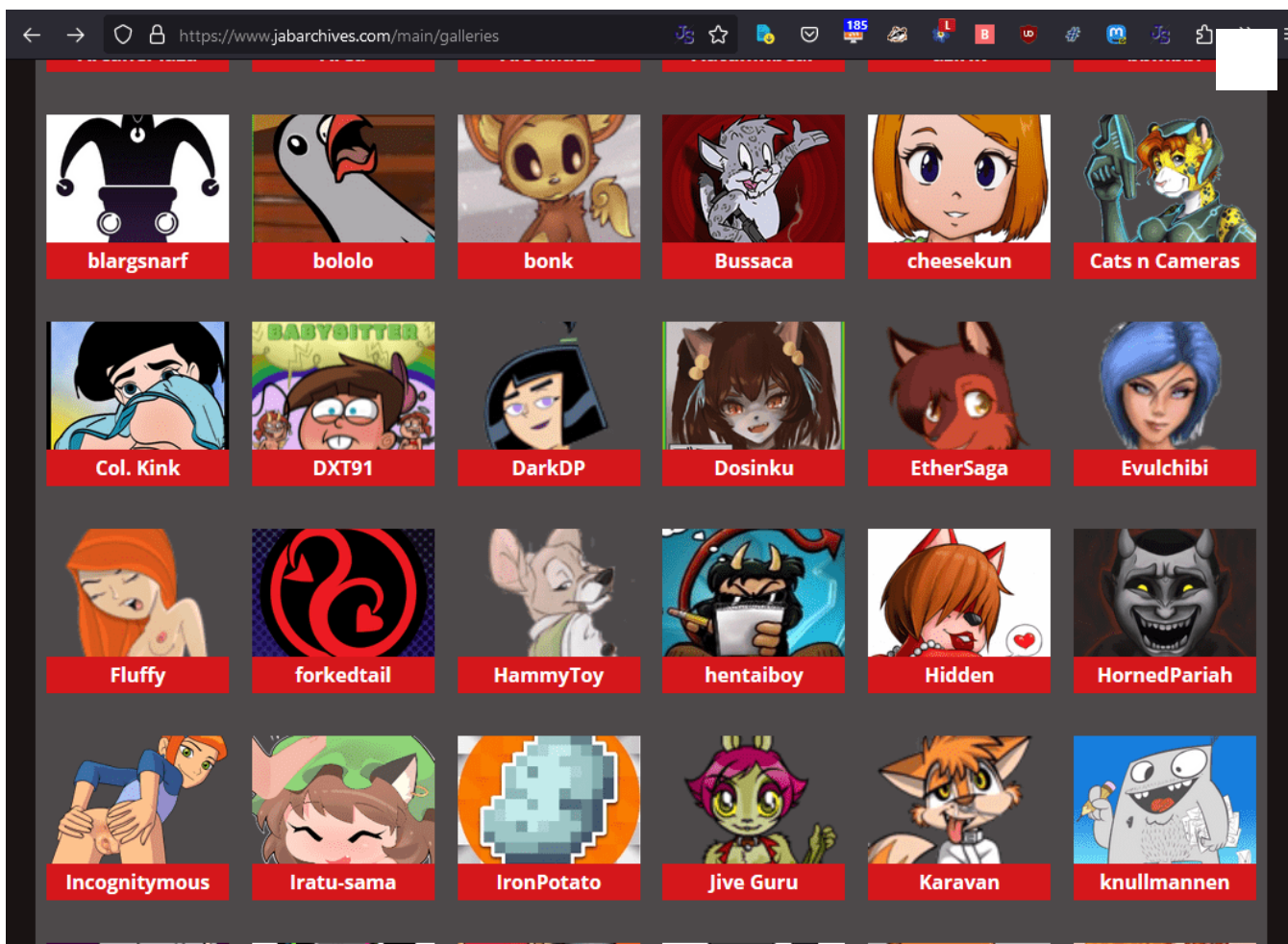
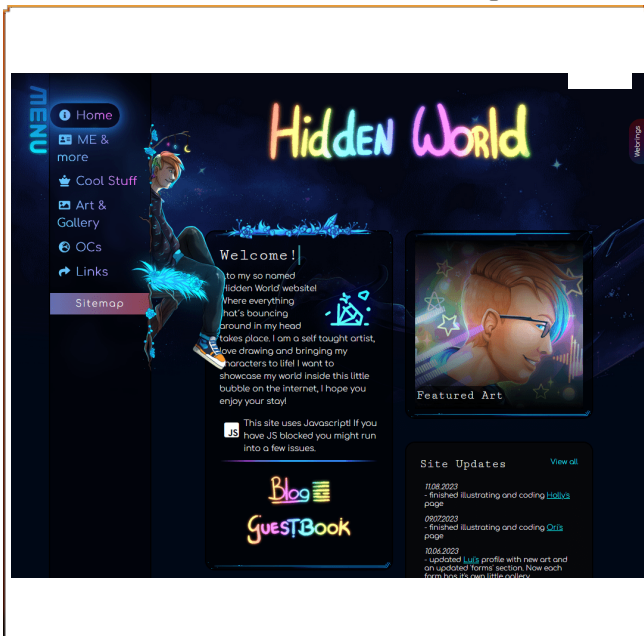
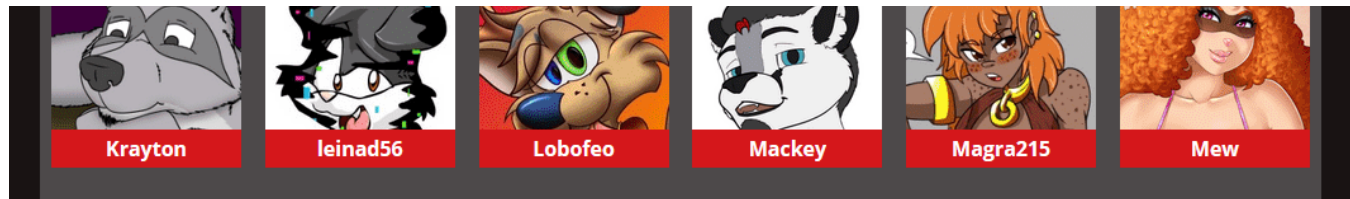


### Why Use Webrings in 2023?

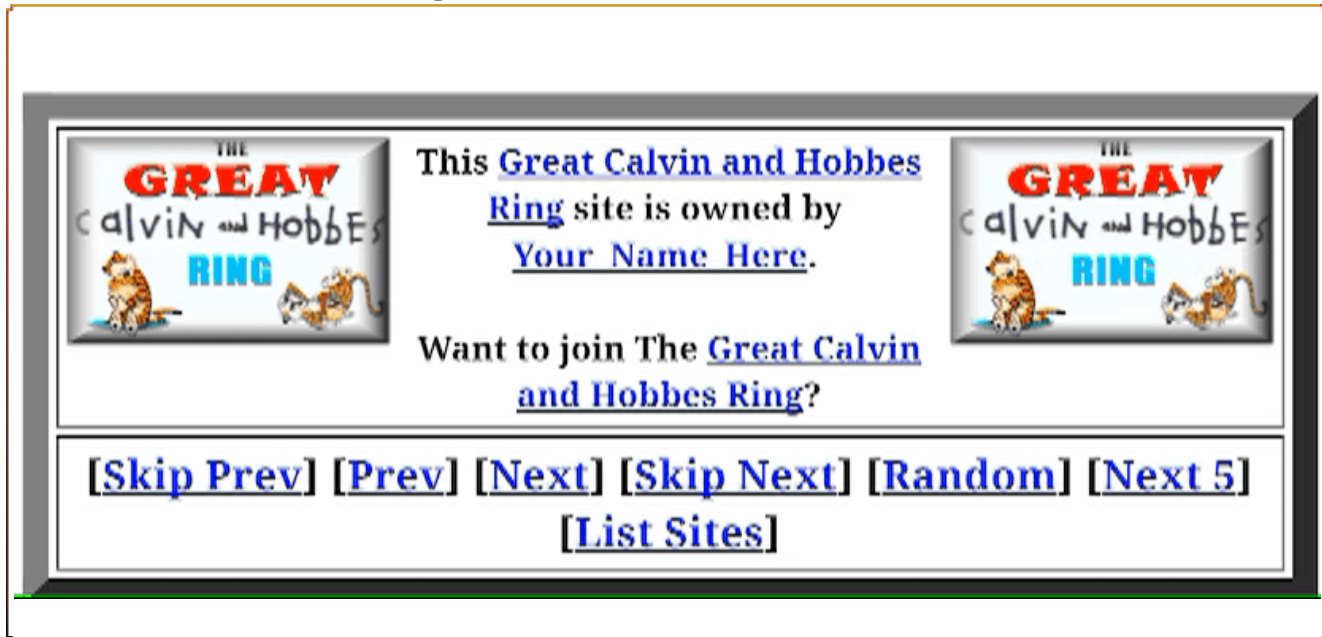
Finding adult artists is easy these days, but finding artists with their own websites is actually pretty hard. Most people only use social media or art sites like DeviantArt or FurAffinity. Search engines can't tell the difference between a hand-crafted personal site and a social media page. It's all HTML to a computer. I love personal websites. You see so much more of a person's creativity in them.





## How to Use Webrings in 2023

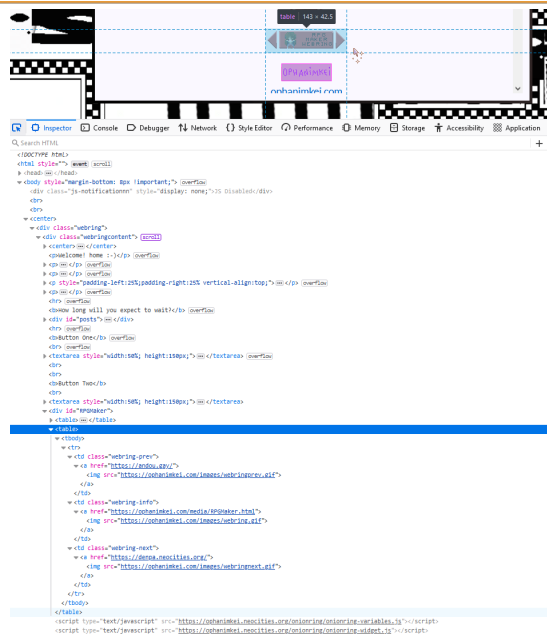
But then I remembered something. Back before dinosaurs walked the earth and Google didn't exist, fan websites used to link to each other using Web-Rings. That way if you found one website about a subject, you could find more.



Sure enough I did a little looking around and found some new webrings people have been creating for adult art. Apparently a lot of adult artists are tired of their stuff getting hidden or deleted off social media all the time. Many of these web-rings revolve around Neocities, which is delightful. So today I added 3 webrings to my page to hopefully make it easier for other people to find me. Neo Creatives Webring, The May December Club, and the RPG Maker Webring (which does allow adult content)

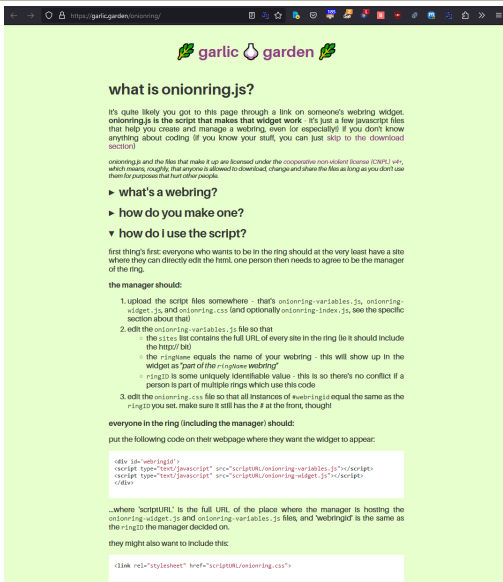
## Doing it the Right Way / Hard Way

And of course, me being me, I couldn't simply copy-paste their link code and call it a day. Nope I had to go and *look* at the code, then recoil in horror at the fragile sequential script tags, global variables, and table tags. So... many... tables... You're only adding 3 tiny pictures guys, what are you doing??

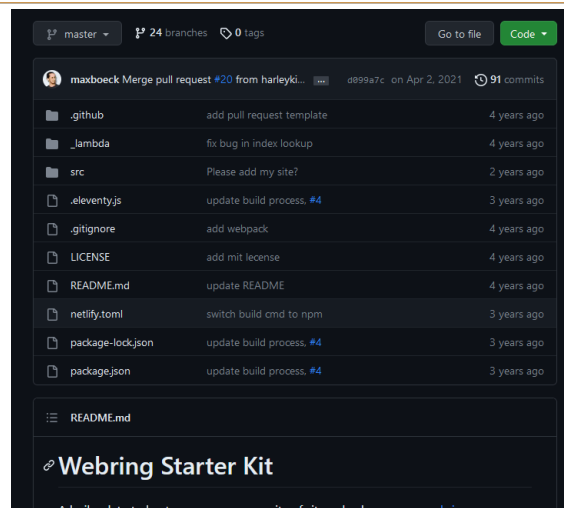


I'm not picking on anybody here. They *all* do this, because they're all using the same code. A web-ring system called [OnionRing.js](#) which is actually the simplest and least-terrible one available at the moment... if you can believe it. The other 3 web-ring systems I looked at all required a GitHub account, a Netlify account, and a PHD in bloated code. Why do you need to sign up to 2 other websites and their dependencies for something *this* simple!? Suddenly [OnionRing.js](#) doesn't seem so bad in comparison.

### Two scripts and global variables... not ideal



### Two accounts depending on two external websites and ALL THESE FILES... all just to link to a simple list of websites??



In the `head` of each part of their page to apply some basic styling to the widget again, `scriptable` should be the URL where the `scriptable.css` file is hosted - it's probably be the same as the one used in the widget source.  
that's all that's as much as you need to do to get it working.

Inspired by posts from [Tatiana Mac](#) and [Charlie Owen](#).

Uses [Eleventy](#) and [Netlify](#) to build a central directory for member sites. People can link to `/prev`, `/random` and `/next` and be redirected to members of the ring.

[Explanatory Blog Post](#)

You might think I'm over-reacting, but a web-ring is *really* not that complicated. It's basically just a list of websites...

```
//the full URLs of all the sites in the ring
var sites = [
  'https://100years.neocities.org/',
  'https://rudytues.day',
  'https://eulogaic.moe/',
  'https://vsitante.neocities.org/',
  'https://sodium-amytal.neocities.org/',
  'https://shirakotoko.moe/main.html',
  'https://www.kradeelav.com',
  'https://gaylie.neocities.org/',
  'https://mappapapa.neocities.org/',
  'https://feltyhatter.neocities.org/',
  'https://scumsuck.com/',
  'https://ninniearts.neocities.org/'
];
```

... and a few links.



Those 5 links can be almost 5 lines of HTML. And navigating a list of text is trivial in JavaScript.

```
//this is the code that displays the widget - EDIT THIS if you want to change the structure
tag.insertAdjacentHTML('afterbegin', `
  <div class='webring'>
    <a href='${sites[previousIndex]}'></a>
```

```

    <a href='` + indexPage + ` '></a>
    <a href='${sites[nextIndex]}'></a>
    <div class="links">
      ${randomText}
      ${indexText}
    </div>
  </div>
);

```

```

// after loading the file, get the list of websites from it.
function fileLoaded( contents ) {
  // output variable
  var website_ary = [];
  // look for the website list inside the text file
  var varAt = contents.indexOf("var sites");
  if ( varAt > -1 ) {
    var arrayStartAt = contents.indexOf("[", varAt);
    if ( arrayStartAt > -1 ) {
      var arrayEndAt = contents.indexOf("]", arrayStartAt);
      if ( arrayEndAt > -1 ) {
        var arrayContents = contents.substring( arrayStartAt, arrayEndAt+1 );
        // convert all to ""
        arrayContents = arrayContents.split("\n").join("");
        // remove spaces
        arrayContents = arrayContents.split(" ").join("");
        // remove tabs
        arrayContents = arrayContents.split("\t").join("");
        // remove carriage returns
        arrayContents = arrayContents.split("\r").join("");
        arrayContents = arrayContents.split("\n").join("");
        // fix extra comma before closing bracket
        arrayContents = arrayContents.split(",").join("");
        // convert the text into an actual array
        // JSON can fail, so wrap it to prevent it from stopping this script
        try {
          var website_ary = JSON.parse( arrayContents );
        } catch (err) {
          // if there was an error, then output an empty array
          console.log( "JSON parsing failed. Using an empty website list instead." );
          var website_ary = [];
        }
        // output
        resolve( website_ary );
      } // found: [
    } // found: [
  } // found: "var sites"
} // fileLoaded()

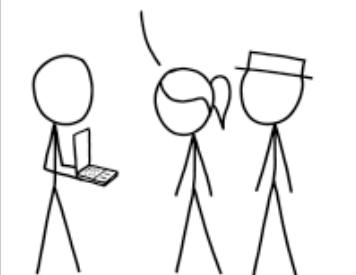
```

That doesn't mean my re-write was short. Just loading a text file takes quite a lot of lines in JavaScript. And since they were not using JSON I have to use a bunch of `indexOf` commands to scan through the text, find the `var sites =` line, find the opening bracket, find the closing bracket, remove all the whitespace and automatically correct any syntax mistakes they made while hand-writing these files, and finally feed the list into JSON to turn it into a normal array. That was most of the code I wrote. The rest is just grabbing an HTML tag with a certain ID and injecting the HTML.

CHECK IT OUT—I MADE A FULLY AUTOMATED DATA PIPELINE THAT COLLECTS AND PROCESSES ALL THE INFORMATION WE NEED.



IS IT A GIANT HOUSE OF CARDS BUILT FROM RANDOM SCRIPTS THAT WILL ALL COMPLETELY COLLAPSE THE MOMENT ANY INPUT DOES ANYTHING WEIRD?



IT... MIGHT NOT BE.

( I GUESS THAT'S SOMETHING WHOOPS, JUST COLLAPSED. HANG ON, I CAN PATCH IT.



And then there's CORS which stands for **Cross-Origin Resource Sharing**. Which actually means the opposite. Browsers don't let JavaScript load stuff from other websites, because advertisers abused this feature so badly that now nobody else gets to do it so I have to

advertisers abused this feature so badly that now nobody else gets to do it... so I had to download all the files containing the website lists ahead of time, upload them to my own website, and *then* read them. But I suppose that's one less external dependency for my website to rely on.

### Downloading stuff just so I can upload it again

```
build_website.cmd x
1
2
3 cd "D:\www\web2.0\src\html"
4 hugo.exe
5
6
7
8 ECHO Updating WebRings
9 wget.exe "https://neocreatives.neocities.org/onionring-variables.js" --directory-prefix "./public/webrings/neocretives/"
10 wget.exe "https://ophanimkei.neocities.org/onionring/onionring-variables.js" --directory-prefix "./public/webrings/rpgmaker/"
11 wget.exe "https://maydecember.neocities.org/onionring-variables.js" --directory-prefix "./public/webrings/maydecemberclub/"
12
```

Anyway it works. Now I have clean reliable code with no global variables and much simpler HTML. I *could* have just pasted their code in 2 minutes and it would have worked fine, tables and all. I just happen to be picky about this stuff.

