The SVG `path` Syntax: An Illustrated Guide

Author
Chris Coyier

Last Updated
Dec 31, 2018

The `<path>` element in SVG is the ultimate drawing element. It can draw anything! I've heard that under the hood all the other drawing elements ultimately use path anyway. The path element takes a single attribute to describe what it draws: the `d` attribute. The value it has is a mini syntax all to itself. It can look pretty indecipherable. It's a ton of numbers and letters smashed together into a long string. Like anything computers, there is a reason to the rhyme. I'm no expert here, but I thought it would be fun to dig into.

Here's an example of a medium-complexity path, I'd say:

```svg
<path d="M213.1,6.7c-32.4-14.4-73.7,0-88.1,30.6C110.6,4.9,67.5-9.5,36.9,6.7C2.
  C33.3,145.1,67.5,170.3,125,217c59.3-46.7,93.5-71.9,111.5-106.1C263.4,64.2,24
```

We could reformat it to start making sense of it (still valid code):

```svg
<path d="
  M 213.1,6.7
  c -32.4-14.4-73.7,0-88.1,30.6
  C 110.6,4.9,67.5-9.5,36.9,6.7
  C 2.8,22.9-13.4,62.4,13.5,110.9
  C 33.3,145.1,67.5,170.3,125,217
  c 59.3-46.7,93.5-71.9,111.5-106.1
  C 263.4,64.2,247.2,22.9,213.1,6.7
  z" />
```

The letters are **commands**. The numbers are **passing values** to those commands. All the commas are optional (they could be spaces).

For example, that first command is `M. M` says, in a metaphorical sense, *pick up the pen and move it to the exact location* `213.1, 6.7`. Don't draw anything just yet, just move the location of the Pen. So that if other commands do drawing, it now starts at this location.

`M` is just one of many path commands. There are 18 of them by my count.

Many (but not all of them) come in a pair. There is an UPPERCASE and a lowercase version. The UPPERCASE version is the **absolute** version and the lowercase is the **relative** version. Let's keep using M as an example:

- `M 100,100` means "Pick up the pen and move it to the exact coordinates 100,100"
- `m 100,100` means "Move the Pen 100 down and 100 right from wherever you currently are."

Many commands have that same setup. The lowercase version factors in where the "pen" currently is.

Let's look at two absolute commands:

Followed by a relative command:

Just like the `M` and `m` commands, `L` and `l` take two numbers: either absolute or relative coordinates. There are four other commands that are essentially simpler versions of the line commands. They also draw lines, but only take one value: horizontal or vertical. When we used `l 25,0` we could have used `h 25` which means "from where the pen currently is, draw 25 to the right. More succinct, I suppose. It's bigger brother `H`, as we could guess, means to draw to the exact horizontal coordinate 25. `V` and `v` move vertically absolutely and relatively as you'd surely guess.

Check out this Chris Nager demo in which he draws a cross in an extremely tiny amount of code, thanks to relative coordinate drawing:

Embedded Pen Here

See the very last character Chris used there? `Z`. `Z` (or `z`, it doesn't matter) "closes" the path. Like any other command, it's optional. It's a cheap n' easy way to draw a straight line directly back to the last place the "pen" was set down (probably the last `M` or `m` command). It saves you from having to repeat that first location and using a line command to get back there.

Let's look at the commands we've covered so far.

M x,y	Move to the absolute coordinates x,y
m x,y	Move to the right x and down y (or left and up if negative values)
L x,y	Draw a straight line to the absolute coordinates x,y
l x,y	Draw a straight line to a point that is relatively right x and down y (or left and up if negative values)
H x	Draw a line horizontally to the exact coordinate x
h x	Draw a line horizontally relatively to the right x (or to the left if a negative value)
V y	Draw a line vertically to the exact coordinate y
v y	Draw a line vertically relatively down y (or up if a negative value)
Z	Draw a straight line back to the start of the path

(or
z
)

So far we've looked at only straight lines. Path is a perfectly acceptable element and syntax for that, although it could be argued that elements like `<polygon>` might have an even easier syntax for straight-line shapes, if slightly more limited.

The superpower of path is curves! There are quite a few different types.

Remember the first bit of example code we looked at used a lot of `C` and `c` commands. Those are "Bezier Curves" and require more data do their thing.

The C command takes three points. The first two points define the location of two bezier curve handles. Perhaps that concept is familiar from a tool like the Pen tool in Adobe Illustrator:

The last of the three points is the end of the curve. The point where the curve should finish up. Here's an illustration:

The lowercase `c` command is exactly the same, except all three points use relative values.

The `S` (or `s`) command is buddies with the C commands in that it only requires *two* points because it assumes that the first bezier point is a reflection of the last bezier point from the last S or C command.

The `Q` command is one of the easier ones as it only requires two points. The bezier point it wants is a "Quadratic" curve control point. It's as if both the starting and ending point share a single point for where their control handle end.

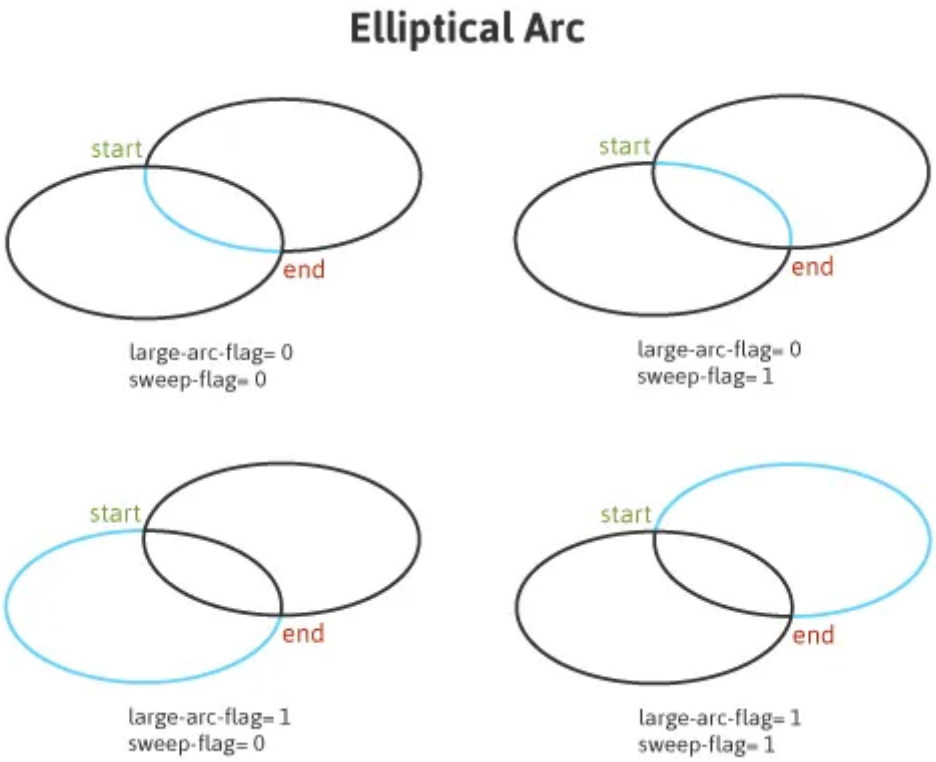
We might as well cover `T` at the same time. It's buddies with `Q` just like `S` is with `C`. When T comes after a Q, the control point is assumed to be a reflection of the previous one, so you only need to provide the final point.

The `A` command is probably the most complicated. Or the require the most data, at least. You give it information defining an oval's width, height, and how that oval is

rotated, along with the end point. Then a bit more information about which path along that oval you expect the path to take. From MDN (https://developer.mozilla.org/en-US/docs/Web/SVG/Tutorial/Paths) :

> *"there are two possible ellipses for the path to travel around and two different possible paths on both ellipses, giving four possible paths. The first argument is the large-arc-flag. It simply determines if the arc should be greater than or less than 180 degrees; in the end, this flag determines which direction the arc will travel around a given circle. The second argument is the sweep-flag. It determines if the arc should begin moving at negative angles or positive ones, which essentially picks which of the two circles you will travel around."*

Joni Trythall's graphic explaining A from her article on SVG paths (https://www.sitepoint.com/closer-look-svg-path-data/) is pretty clear:



Here's written explanations of those curve commands.

c cX1,cY1 cX2,cY2 eX,eY	Draw a bezier curve based on **two** bezier control points and end at specified coordinates

c	Same with all relative values
S cX2,cY2 eX,eY	Basically a C command that assumes the first bezier control point is a reflection of the last bezier point used in the previous S or C command
s	Same with all relative values
Q cX,cY eX,eY	Draw a bezier curve based a **single** bezier control point and end at specified coordinates
q	Same with all relative values
T eX,eY	Basically a Q command that assumes the first bezier control point is a reflection of the last bezier point used in the previous Q or T command
t	Same with all relative values
A rX,rY rotation, arc, sweep, eX,eY	Draw an arc that is based on the curve an oval makes. First define the width and height of the oval. Then the rotation of the oval. Along with the end point, this makes two possible ovals. So the arc and sweep are either 0 or 1 and determine which oval and which path it will take.
a	Same with relative values for eX,eY

Wanna see a bunch of examples? OK:

Embedded Pen Here

If you're looking in a recently-released Blink-based browser and you have a mouse, you'll see some hover animations! Turns out you can set path data right in CSS now. For example...

```svg
<svg viewBox="0 0 10 10">
  <path d="M2,5 C2,8 8,8 8,5" />
</svg>
```

```css
svg:hover path {
  transition: d 0.2s;
  d: path("M2,5 C2,2 8,2 8,5");
}
```

Wanna know more about SVG? It's seriously cool I promise. I wrote a whole book on it. It's called Practical SVG (https://abookapart.com/products/practical-svg) and it's not very expensive.