Inkscape - Part 94

After the JavaScript detour of the past few months, it's time to head back into the realms of pure Inkscape. Before we dive headlong into the next topic, however, I'm going to spend this month and next tidying up a few loose ends and updating you on some recent Inkscape news items.

To begin with, let's finish up with a little more polish on the JavaScript-based transform animation we've been building up through the last few articles. If you followed along, you'll have ended up with four blocks of code that all look very similar, for each of skewX, skewY, scaleX and scaleY (top right).

All that changes between them are the variable names, and the only result we use from this code is the single value we create at the end. This block is an excellent candidate for splitting out into a single function that gets called for each of the four transform parameters. Reorganising code to make it simpler, clearer, or more efficient, is referred to as "refactoring". To refactor this code, therefore, the first step is to copy it into a function of its own, and return the value we're interested in. We'll add the new function to the very end of the JS file (bottom right).

```
const skewXDur = props.skewXDuration;
const skewXRange = props.skewXMax - props.skewXMin;
let skewXPosition = (runningTime % skewXDur) / skewXDur;
const skewXDirection = runningTime % (skewXDur * 2);
if (skewXDirection > skewXDur) {
  skewXPosition = 1 - skewXPosition;
}
const skewXAmount = (skewXRange * skewXPosition) + props.skewXMin;
```

We're going to use this function to replace blocks of code that deal with multiple different properties, so it makes sense to also tidy up the variable names in the function to make them more generic. By passing in the relevant duration, min, and max, from the properties – and also passing the current running time that was supplied to the parent animate() function – we can simplify it to this general purpose function (below).

```
function getAnimAmount(dur, min, max, runningTime) {
  const range = max - min;
  let position = (runningTime % dur) / dur;
  const direction = runningTime % (dur * 2);
  if ( direction > dur) {
    position = 1 - position;
  }
  const amount = (range * position) + min;

  return amount;
}
```

Now we can replace the original four blocks with something much simpler. Here's the line to use for

```
function getAnimAmount() {
  const skewXDur = props.skewXDuration;
  const skewXRange = props.skewXMax - props.skewXMin;
  let skewXPosition = (runningTime % skewXDur) / skewXDur;
  const skewXDirection = runningTime % (skewXDur * 2);
  if (skewXDirection > skewXDur) {
    skewXPosition = 1 - skewXPosition;
  }
  const skewXAmount = (skewXRange * skewXPosition) + props.skewXMin;

  return skewXAmount;
}
```

skewX – I'll leave it to you to work out what to do with the others. Note that I've split this across multiple lines to squeeze it into a single column in this article, though in a real file I would probably put all of this in a single line of code:

```
const skewXAmount =
  getAnimAmount(
    props.skewXDuration,
    props.skewXMin,
    props.skewXMax,
    runningTime
  );
```

It almost goes without saying, but once you've replaced all four blocks of code you should save and reload your page. If everything was correct you should see no change in the animation.

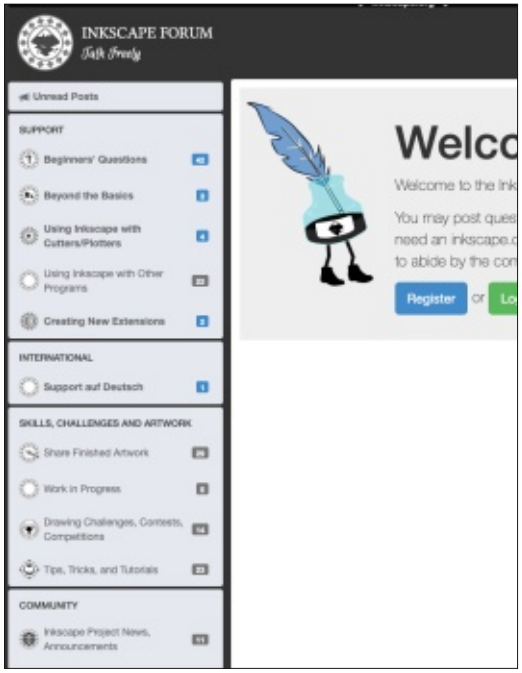
You should, however, be able to see at a glance how much shorter and simpler this change makes the animate() function. It's gone from being a mathematically heavy chunk of code to a much simpler series of lines that just set up some constants, applies them to the transform() attribute, then calls itself for the next iteration. If you dive further into coding you should always try to keep an eye out for repeated blocks of code that can be refactored into a single function.

With that, I'm going to draw a line under this part of the series – at least metaphorically speaking, although you should now have enough understanding of SVG and JavaScript to be able to draw a literal line, if you wish.

Forum News

I'm sad to announce that the long-standing semi-official forum for Inkscape, "inkscapeforum.com", is no longer around. It hasn't been actively supported by its administrator for a number of years, but had still been the go-to place for the English-speaking Inkscape community. Despite the efforts of a number of users and developers – myself included – to transfer the forum into the stewardship of the Inkscape project, the admin did not provide either a backup of the database, nor enough access for us to obtain it ourselves. A couple of months ago the site simply disappeared, and has not returned.

Fortunately the lack of communication from the admin acted as a warning, and an officially-official forum was already up and running when the old one vanished – thanks mostly to the sterling efforts of Martin 'Doctormo' Owens, a long-standing contributor to the Inkscape project. With its main 'competitor' having self-destructed, the official forum has grown in popularity, and should now be seen as the best place to go if you have an Inkscape question, or just want to show off your Inkscape-created images. The new forum is linked from the main Inkscape website, or you can access it directly at the following address:
https://inkscape.org/forums/



But what of the many years' worth of hints, tips, tutorials, questions and answers that had accrued on the old forum? Thankfully it's not all lost: the Inkscape website is hosting a read-only backup which mostly works, though some threads don't seem to behave quite as they should.
https://alpha.inkscape.org/vectors/www.inkscapeforum.com/

Alternatively the Wayback Machine at the Internet Archive site also has a snapshot of the old forum though, again, it's not perfect.
https://web.archive.org/web/20190910200439/http://www.inkscapeforum.com/

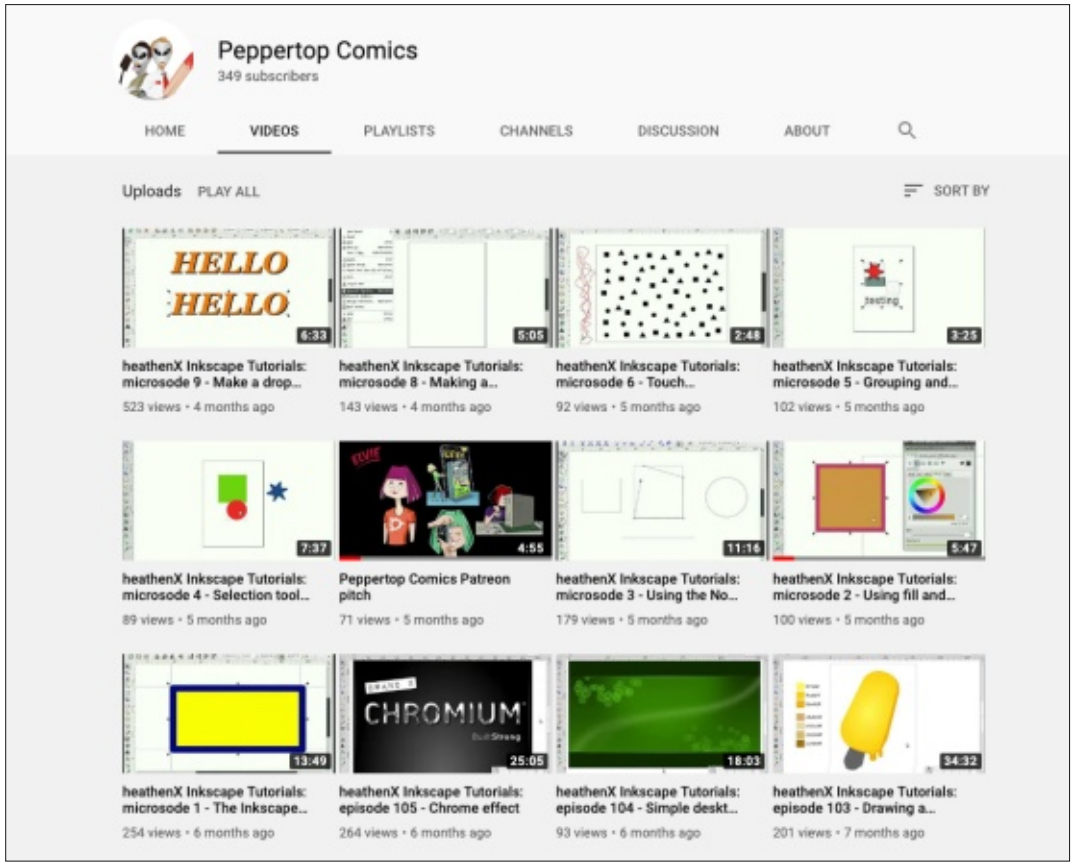
Hopefully between these two resources the bulk of the useful content from the site has been preserved.

One very useful page from the old forum which hasn't been completely captured is the index of my Full Circle Magazine articles. The last snapshot on the Wayback Machine dates to 2017 and, although the page on the Inkscape site is more up-to-date, I've nevertheless copied the content to

my own site, where I can continue to update it as new articles are published. You can find it here: http://www.peppertop.com/blog/?p=1563

Video Tutorials

When I was first starting to learn how to use Inkscape, a valuable resource was a series of tutorial videos created by 'Heathenx' and Richard Querin. Unfortunately, their website has long-since disappeared, but I was able to get a current email address for Heathenx and contacted him about re-posting the videos online. I'm delighted to report that he was extremely helpful, and happy for them to be uploaded to Youtube. There are over a hundred videos in total, most of which are still relevant to the current Inkscape release. I uploaded them over a number of months, adding my own notes to clarify any tweaks or changes that are required for recent Inkscape versions. I urge you to take a look at them, as they provide a great example of the many and varied effects that can be produced with Inkscape if you have a little artistic flair.



https://www.youtube.com/c/PeppertopComics

A Shameless Plug

While we're on the topic of Inkscape resources I've posted online, I'm going to take this opportunity for a spot of blatant self-promotion. Long-term readers of this column (or those who read the byline at the end of an article) will know that I create comics and cartoon strips using Inkscape (with the help of a far more artistically talented friend). We've been working together in our spare time for over 25 years now, but it's the last decade that has been the most interesting as far as this column is concerned. During that time, we've used Inkscape in the creation of over 250 strips which can be freely viewed on our website. To help encourage new Inkscape users, we've made the vast majority of our source files available for download as well, so you can see how we put things together or modify the cartoons to suit your own needs.

Given that you're reading a Linux magazine, I'd like to draw particular attention to our "Elvie" comic strip. This appeared in every edition of the much-missed Linux Voice magazine, and has continued every month in Linux Pro Magazine (just "Linux Magazine" outside North America). Thanks to the kind generosity of the editors and publishers, we've been able to release these strips under a very liberal license (Creative Commons BY-SA), and have a Git repository with our Inkscape and MyPaint source files. We've even gone so far as to use fonts that are under liberal licenses, so that every single part of these cartoons is as free as possible.

Aside from Elvie, our other cartoons have appeared in magazines and newspapers, and even in an exhibition at the National Media Museum in the UK. They can all be viewed and downloaded from our website:

http://www.peppertop.com

If you want to support us in the work we do to promote Inkscape – or if you want to help in our goal to raise enough funds to relicense some of our earlier cartoons as BY-SA – we also have the near-obligatory Patreon page: https://www.patreon.com/peppertop

Inkscape v1.0 Beta

In classic 'saving the best for last' tradition, the final item in this hodgepodge of an article is the announcement of the long awaited release of Inkscape v1.0. Well, nearly.

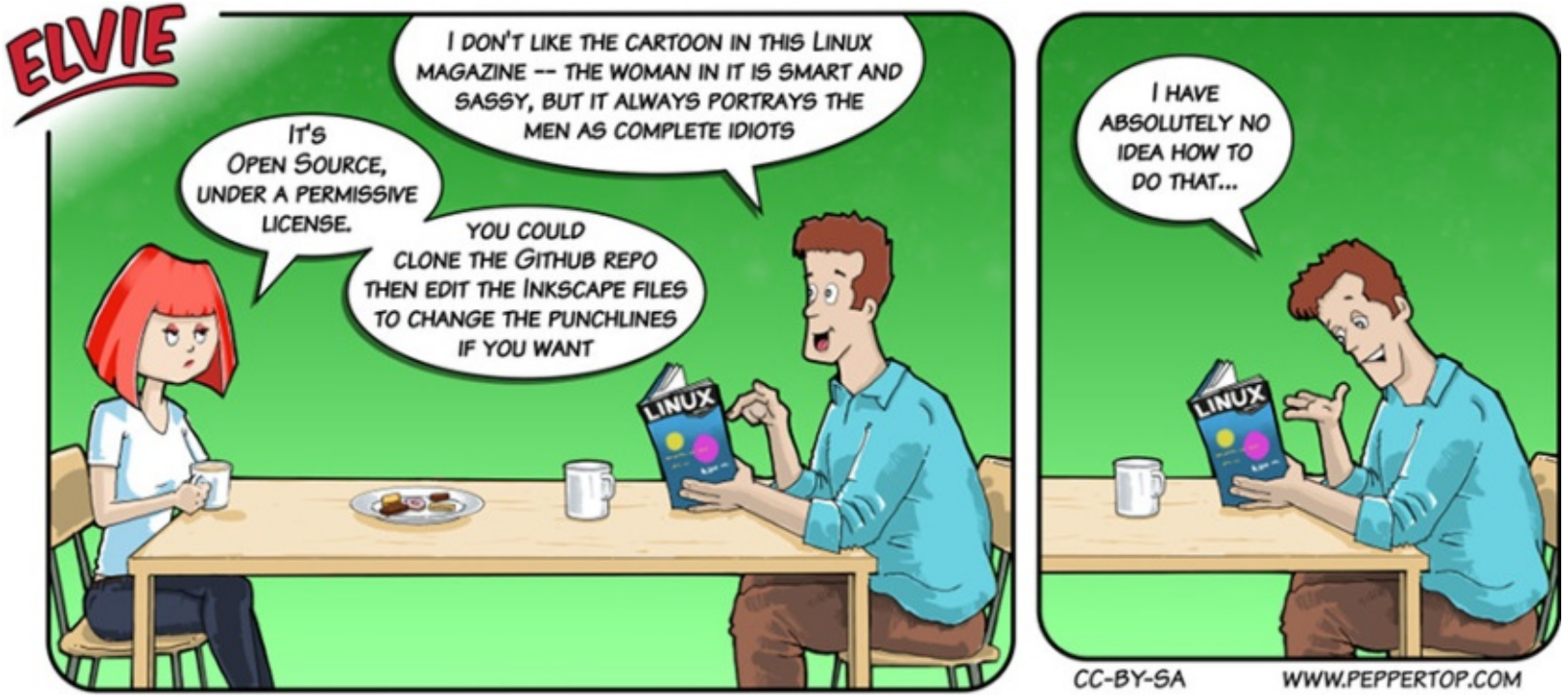
At the time of writing, the second beta of v1.0 has been released, and is available via the download link on the Inkscape website (https://inkscape.org). There's no definite date for the final release, and I don't know if there will be any other beta releases in the meantime, but what's there already feels pretty stable and polished. I encourage readers to give the beta a try, and report any issues they find to the Inkscape bugtracker. Note that the project is no longer using Launchpad to track issues, having switched to GitLab some time ago. If you do want to file a bug report, or view those that have already been filed, there's a page on the Inkscape site that will redirect you to the correct location:

https://inkscape.org/report

A fond farewell... until next month

I had originally intended for this to be my last Inkscape column, having covered just about every feature the program has to offer during the past 8 years. With the imminent arrival of v1.0, however, I've decided to stick around to introduce the new features and important changes. Initially this coverage will necessarily be based on the beta versions, but it's unlikely that there will be significant UI changes before the full release so I'm sure anything I write will still be relevant in a few months' time.

Next month, I'll be detailing a couple of minor features in Inkscape that have managed to slip through the cracks in previous instalments, to fully 'clear the decks' before diving into the exciting new features of v1.0.

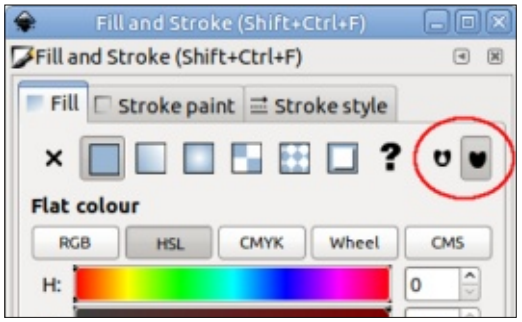


Mark uses Inkscape to create three webcomics, 'The Greys', 'Monsters, Inked' and 'Elvie', which can all be found at http://www.peppertop.com/

Inkscape - Part 95

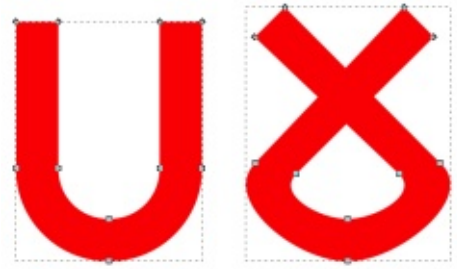
This month and next, I'm going to cover a couple of Inkscape features that have fallen through the gaps in my previous coverage. The first of these, disgracefully, dates back to the 4th instalment of this series, way back in 2012! In FCM#64, when describing the options in the Fill and Stroke dialog, I wrote: "*Also being put off for later articles are the Unset Paint button that looks like a question mark, and the two splodges on the right of the Fill tab.*"

The Unset Paint button was eventually described in part 30 (FCM issue #90), when I covered its use with clones. But what of the "two splodges on the right of the Fill tab"? For clarity, let's first take a look at a screenshot of the Fill and Stroke dialog, in which I've circled the splodges.



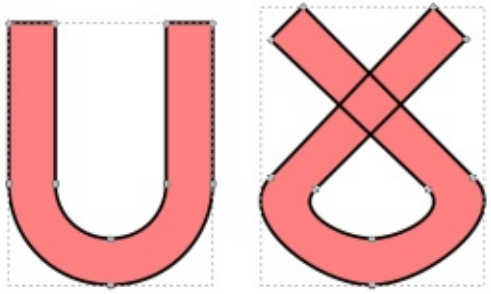
To understand what effect these splodges – which are actually a pair of mutually exclusive radio buttons – have on your objects, we'll need a path to test with. Not just any path will do, however. These buttons have an effect only on paths that are either self-intersecting, or which contain sub-paths. Let's deal with the former to start with: what exactly do I mean by 'self-intersecting'?

Consider the paths below. The one on the left is just a simple U-shaped filled path. Nothing special, no sneaky tricks. Just 10 nodes forming a shape with a couple of arms. The shape on the right is just a duplicate of the first one, with the end of the left arm moved to the right, and the end of the right arm moved to the left, so the arms cross each other. No nodes were added or removed, they were just moved around a little.



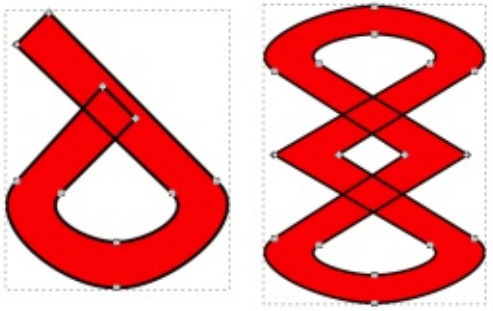
The second shape is a self-intersecting path. It's a closed path that is drawn such that it crosses itself, although the word 'crosses' carries a misleading implication in this case. Remember that Inkscape is a 2D drawing program, with no notion of the third dimension. So when making this shape I didn't really move one arm 'over' the other, as you would do when making the same shape with a piece of string. Rather I moved it 'through' the other, in the same 2D plane. Neither arm is on top of the other. Neither arm 'crosses' the other. Instead the resultant shape simply intersects with itself.

Turning on the stroke and reducing the fill opacity makes this a little clearer:



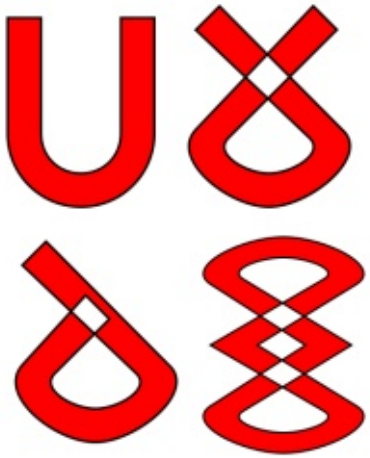
Notice how the stroke is visible for both arms. If one crossed over the other you might expect the stroke of the arm on the bottom to be obscured by the one that's on top, but that's not how things work in the 2D graphics world. You'd also expect the colour of the fill to be a little darker where the two translucent arms cross over, but that's not the case either.

It's also important to note that the arms don't have to completely pass through each other in order to be a self intersecting shape, and that the shape could self-intersect multiple times, as demonstrated in these variations on the theme:



Now we're all clear on what a self-intersecting path actually is, let's look at how they're affected

by the splodgy radio buttons. The first thing to note is that the images so far have all been taken with the second button selected – the one that's completely filled in. Switch to the first button, and we get this:



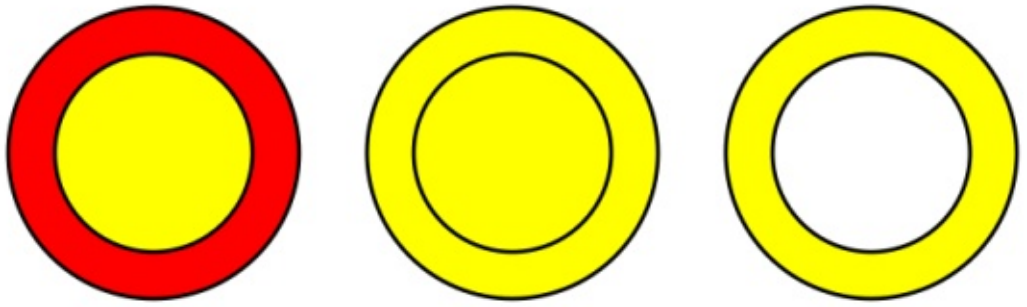
The basic U-shaped path remains unaffected. It's not a self-intersecting path, and it doesn't contain any sub-paths, so the radio buttons have no effect. All the others, however, have no fill rendered in the intersecting parts. The stroke is still drawn, but the fill isn't.

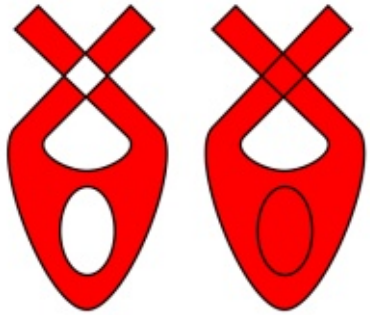
This behaviour is a little confusing at first. Surely the intersecting parts are inside the shape, so they should still be filled, right? Unfortunately our intuitive idea of in-and-out doesn't always apply in the world of computer graphics. Instead there are different rules or heuristics for determining whether a point is inside or outside a shape – and the splodgy buttons are used to switch between a couple of those rules. Let's take a look at paths with sub-paths to try to explain why there's a need for this choice.

When you look at the self-intersecting shapes above, it's pretty obvious that the fully-filled option (second button) is the right choice most of the time. Imagine trying to draw a figure-of-eight, only to find that the part where the shape crosses itself is unfilled. Definitely better to use the second option, so we'll ensure that's selected. Now we're going to draw a circle with a hole in it. There are a few ways to achieve this but, to make a point, I'm going to use a very specific method:

- Draw a circle for the outside.
- Draw a smaller circle for the inside.
- Path > Combine to create a complex path, made up of two sub-paths.

In this image (above) you can see the two separate shapes on the left, and the result of combining them in the middle. The 'hole' in the shape is filled – not really what we were after. Switching to the first button, however, gives the expected result, as shown in the version on the right.





Now we've got a simple rule-of-thumb: use the left-hand button for paths with sub-paths, and the right-hand button for self-intersecting paths. But what if we have a path that is both self-intersecting and has a sub-path describing a cut-out? Something like this odd piece of jewelry or alien symbol, shown as it appears with each of the two splodgy radio buttons selected:

On the left we have the problem of no fill in the self-intersecting part, but the hole is fine. With the right-hand button we get a fill in the self-intersecting part, but no hole. Is there any way we can have the best of both options?

The answer is obviously 'yes', but to get there it helps to understand what the two heuristics are that we've been switching between. The first button has a tooltip that reads "any path self-intersections or subpaths create holes in the fill", and is referred to as the "even-odd" rule. The heuristic for this is pretty simple:

- Pick a point in the shape.
- Draw an imaginary straight line from that point out of the shape in any direction.

• Keep a 'crossing' count, which starts at zero.
• Add one to the count each time the line crosses a path or sub-path boundary.
• If the final count is even, the point was outside the shape. If the count is odd, the point was inside the shape.

By performing this for a point in each region of the shape, Inkscape can determine which areas are inside, and therefore need to be filled, and which are outside. But it does result in self-intersections being counted as outside, and left unfilled.

The second button has a tooltip that reads "fill is solid unless a subpath is counterdirectional", and is referred to as the "non-zero" or "winding" rule. This heuristic relies on the fact that each sub-path has a direction associated with it, and works like this:
• Pick a point in the shape.
• Draw an imaginary straight line from that point out of the shape in any direction.
• Keep a 'winding count' which starts at zero.
• Each time the line crosses a path or sub-path, add one to the count if the path is crossing the line from left to right (clockwise from the perspective of the point) or subtract one if the path is crossing from right to left (counter-clockwise).
• If the final count is zero the point is outside the shape, otherwise it is inside.

From this algorithm and the mention of "counterdirectional" in the tooltip – you may have already worked out that using the second splodgy button (non-zero) lets us switch sub-paths between filled and unfilled by reversing the direction of the nodes. Fortunately for us, Inkscape makes this fairly easy:
• Select the composite path and switch to the node tool (F2).
• Select a node in the sub-path.
• Use Path > Reverse to reverse the order of the nodes in the sub-path without changing the shape.

You can see the effect of this approach in the following image. In this case I've also added arrow markers to the paths, to clarify what's happening. The first image is our starting path, created by combining an oval with a self-intersecting shape. You can see from the arrows that the sub-path is running counter-clockwise. In the second image I've reversed the direction of the sub-path, the arrows point the opposite way, and the shape has a hole for the sub-path but not for the self-intersection.

Unfortunately this simple approach doesn't appear to work in the beta of Inkscape v1.0, where the Path > Reverse operation reverses the entire path rather than the selected sub-path. The only option in this case is to use Path > Break Apart to split the shape down into its constituent parts, then use Path > Reverse on just one of those paths, then finally use Path > Combine to re-combine them back into your original complex path. This is a definite step backwards in the workflow, so I will be filing a bug about this on the Inkscape issue tracker.

This path-reversing approach works for simple shapes where the nesting of sub-paths isn't very deep. For more complex arrangements you may need to reverse more than one path to get the effect you want. And if there are also intersections between sub-paths, or self-intersections within them, you may never be able to get just the fill you want, no matter what you try. In those cases you may find it easiest to use the first splodge (even-odd) to fill the entire shape, then create your complex hole arrangement by clipping with a suitably constructed path.

It's worth noting that you might never experience this problem. Punching a hole through a shape is more commonly done using Path > Difference than Path > Combine, and the former usually results in a sub-path going in the right direction. But it's worth knowing about these splodgy buttons and how to work with them, just in case you ever find yourself presented with a path that contains inexplicable holes or fills where it shouldn't.

Inkscape - Part 96

In this instalment I'm going to cover a common requirement that I overlooked when introducing the use of SVG files in a web browser: turning an object in an Inkscape file into a clickable link that loads a different URL.

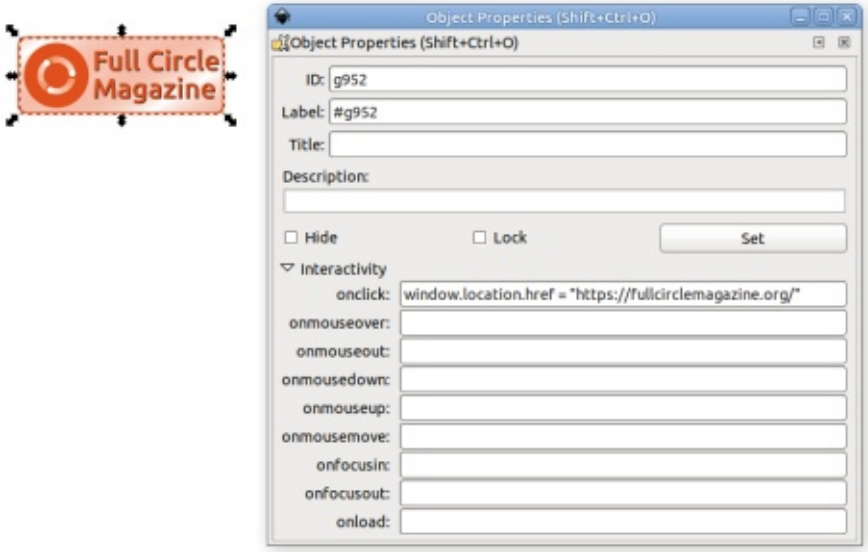
There are a couple of ways of dealing with this: the first is to use JavaScript to respond to the 'click' event that fires when an object is clicked on. I covered various ways to add JavaScript to an Inkscape file across several articles. See FCM #142, #143 and #146 for the specifics. What I didn't describe was how you could use JS to change the URL loaded in the browser.

In the most basic form, where you just want to move to a fixed URL when an object is clicked, you can use the one-line "onclick" field in the Interactivity section of the Object Properties dialog (FCM #142). For example, to make a button that goes to the Full Circle Magazine website, you would do the following:

- Draw your button. Use multiple objects and text as you wish.
- Put all of the button's content into a single group. This is where we'll attach the click handler.
- Right-click on the group, and select "Object Properties".
- Expand the "Interactivity" section, if necessary.
- Add the following JS code to the "onclick" field:

```
window.location.href =
"https://
fullcirclemagazine.org/";
```

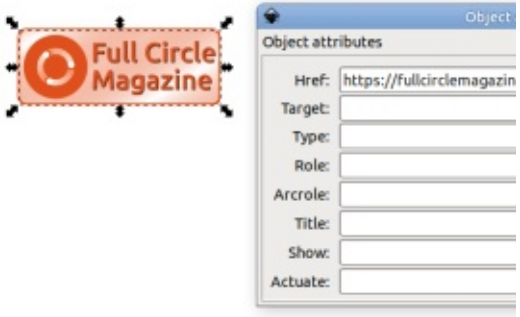
Your button and dialog should look something like that below.



Save the SVG file then load it into a web browser and you should find that clicking the button takes you to the FCM website (or whatever URL you used).

You'll probably have noticed what you don't get for free with this approach: there's no change of style of the button as you hover over it, and the mouse pointer remains as an arrow rather than changing to the "pointing finger" which is usually used to denote a clickable target. Both these shortcomings can be addressed with a little CSS, but that's yet another chunk of code to manually add to your SVG file (FCM #145 will help with this).

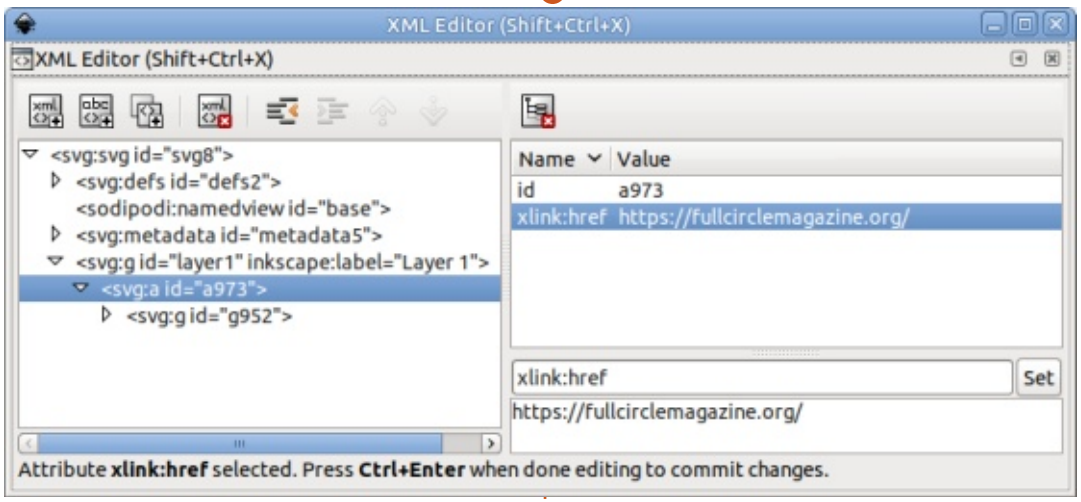
If all you want is a link to another URL, however, there's no need to mess with JavaScript at all (though you may still need some CSS). Inkscape provides a simpler way to turn an object into a clickable link – and it's this part of the UI that I overlooked in my previous articles. All you need to do is to right-click on the object and select "Create Link" to open the generically named "Object attributes" dialog:



That's a lot of fields for a simple link. The reason for this is that Inkscape creates an SVG 1.1 version link, which is actually implemented

via an XML standard called XLink. It dates from the time when the W3C was trying to create a wide ranging collection of XML-based standards, with the idea being that a single file might use elements from across multiple specifications, allowing each spec to focus on doing one thing well. XLink, therefore, is a standard that deals with nothing but links between documents – but in trying to include numerous use-cases for links it has a whole load of optional attributes that most people will never need. Hence all the fields.

The only essential field is the first one, "Href" (an abbreviation of "hyperlink reference"). A better title would have been "URL", "Address" or "Location", but this dialog just uses a capitalized version of the attribute name from the XLink standard. So the "href" attribute used in the XML becomes the awkwardly capitalized "Href". With a URL in this field, save the file and load it into your web browser. You should find that clicking the button takes you to the destination page. Furthermore, you'll get the right sort of pointer as you move your mouse over the button, so that's one less bit of CSS to add to your page.



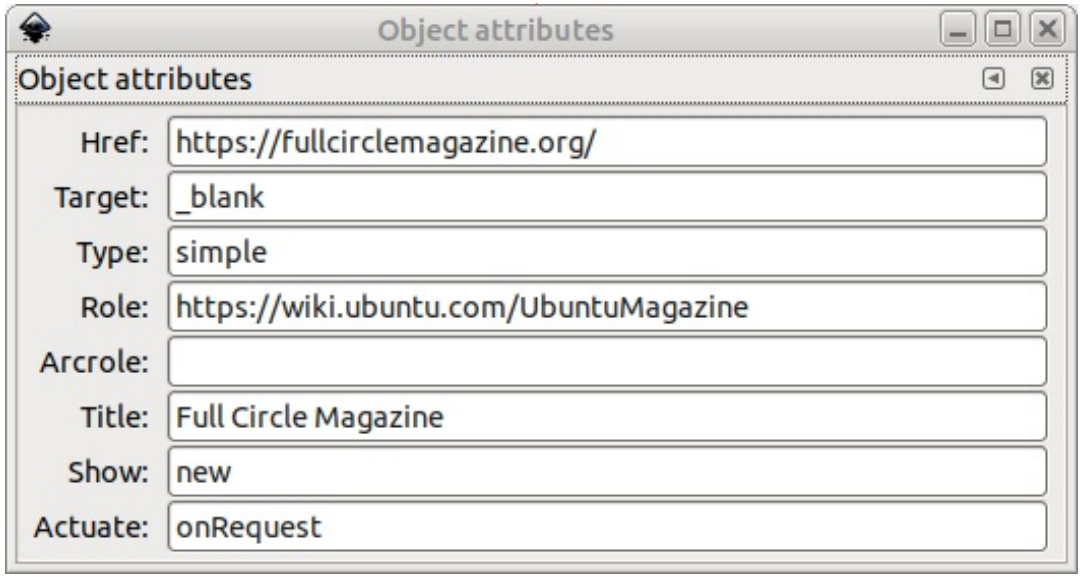
Let's take a look at the XML editor to see what this small change has actually done to your SVG file (shown above).

The first thing to note is that the new attribute hasn't just been added to the existing <g> element. Instead, Inkscape has wrapped the group in an <a> element, and the attribute has been applied to that. Anyone who has written some HTML will be familiar with <a> as the "anchor" element which is used for links in that language. Here we have essentially the same element, but in the "svg" namespace (hence it appears as <svg:a ...> in the XML editor). The URL is added as an attribute in the XLink namespace. If you were to look at the XML file in a text editor, the relevant bits of code would look like this:

```
<a id="a973"
xlink:href="https://
fullcirclemagazine.org/">
  <g id="g952">
    ...
  </g>
</a>
```

Let's fill out most of the remaining fields in the Object Attributes dialog, to try to make our link more fully featured. Having already created the link, you will find that a right-click on the object within the Inkscape window now shows the "Create Link" option as disabled. Instead, a little further up on the context menu, you'll find that the usual "Fill and Stroke…" menu item has been replaced with "Link Properties…" which will open the same dialog (below).

The first thing I'll note here is that you will almost certainly never need to fill out this many fields. "Href" is required, and "Title" is a good idea for accessibility purposes, and also because



desktop browsers will use it to create a tooltip when you mouse over the object. You might need to use the "Target" field, depending on how you want the link to behave, but I'll come on to that shortly.

Let's skip to the "Type" field. This describes the nature of the link, from a specific list of options in the XLink specification. For a normal link to another page (or within the same page), "simple" is all you need. This also happens to be the default behaviour if it's omitted, so you should just leave it blank. The other possible types are all used for more complex linking between and within XML files. If you need to use them then you probably already know about them – and I doubt very much that you'd be using this dialog to edit them anyway.

One of those more advanced types is "arc", which indicates that the link is being used to connect two other resources, identified by the "to" and "from" attributes. A "resource" in these terms is anything that can be identified with a URL, such as a website, a specific page or file, or a named element on that page. As you may have noticed, the "to" and "from" attributes aren't present in the dialog, so you can't actually create a valid arc link through this UI even if you knew why you might want to! For this reason the related "Arcrole" field is also completely useless (if you could create an arc link, this would hold the URL of a resource that describes it).

Based on that description of "Arcrole", you may not be surprised to find that "Role" field is also intended to hold a URL that points to a descriptive resource. In this case it should hold the address of a resource that describes the nature or purpose of the link. Since your web browser doesn't natively do anything with this attribute, however, you may as well omit it.

The "Actuate" field is intended to indicate when the link should be followed. This attribute can only take very specific values but, once again, it's completely ignored by the web browser regardless of what you enter. The easiest option is therefore to leave this blank. The "onRequest" option I've used in my example just means "follow this link when the object is clicked", but that is, once again, the default behaviour anyway.

All that remains are the "Target" and "Show" fields. These attributes actually perform the same purpose, but "target" is part of the SVG spec for the <a> element, whereas "show" is an XLink offering. They affect where in the UI the browser loads the linked resource – whether it replaces the SVG file in the same frame or tab, or opens in a completely new tab or window. The main values to be aware of are as follows (note the underscores before the values for "target" (see table below).

As you might guess from the missing values in the "show" column, there's rarely much need for "_parent" or "_top". The best policy is usually to leave the "target" and "show" attributes empty, so that the behaviour of the browser is purely defined by the user's settings. If you're really sure that you want to open a new tab or

Target	Show	Description
_self	replace	Replace the current SVG file in-place with the resource pointed to by the link. If the file is in an <iframe> or <object> the new content will also be put into the <iframe> or <object>.
_parent		The immediate parent of the SVG file is replaced by the linked content. In the example of an SVG inside an <object> inside an <iframe>, only the content of the <iframe> would be replaced – the <object> would remain.
_top		Replace the current tab or page, regardless of whether the SVG is in a hierarchy of <iframes> or similar. In other words, redirect the whole page.
_blank	new	Open the resource in a new tab or window.

window when the element is clicked, then use "_blank" in the "Target" field. But that's pretty much the only legitimate use of this field for most people.

As you can see, it's possible to enter conflicting values for "Target" and "Show". Experiment indicates that, for Firefox at least, "Target" takes priority. All the more reason to leave the "Show" field blank.

So there you have it: a dialog with eight fields, of which you only really need one (Href), might use two if you want to have a tooltip (Href, Title), or stretch to three if you also want to force links to open in a new tab (Href, Title, Target). The remaining fields should always be left empty, unless you really know what you're doing and are something of an XML/XLink expert. But in that case you're undoubtedly either editing the XML content by hand, or via some other XML-based workflow. In neither case is this dialog likely to be of much use to you.

There's one large elephant in the room, however: the use of XLink at all. As I mentioned earlier, this dialog creates an SVG v1.1 link. But since version 2.0 of the SVG spec there's been no need for XLink. The "href" attribute has been promoted to the SVG standard, together with the "target" attribute. Oddly, however, the "title" attribute has not been promoted, though the "xlink:title" version has been deprecated. The recommendation now is to use a <title> child element instead, which seems a little overkill for a simple tooltip.

With this in mind, an SVG2 link might look something like this:

```
<a id="a973"
  href="https://
fullcirclemagazine.org/"
  target="_blank">

  <title>Full Circle
Magazine</title>

  <g id="g952">
    ...
  </g>
</a>
```

For now – and for the foreseeable future – browsers continue to support the SVG 1.1 approach, so there's no urgency for Inkscape to change what it outputs. Modern browsers will also accept the SVG2 version, though, so perhaps some future release of Inkscape will replace this generic dialog with something more tailored to the task, and will replace the output with an SVG2 version at the same time.

The last thing to note on this topic is that the URL you link to doesn't have to be a separate file. You can also link to a named anchor within the current file. This is particularly useful with the techniques I described for creating named views in parts 79 and 80 (FCM #139, #140). For example, given a named view of "starView", simply creating a link with an href of "#starView" would mean that the image would switch to that view when the object is clicked. A similar effect can be achieved with the full viewBox syntax, using an href of "#svgView(viewBox(0, -250, 250, 500))" for example.

This can be an easy way to introduce interactivity to an SVG file. Consider a slideshow in which each slide is a separate part of the SVG image, and a viewBox is used to show just the first slide by default. By adding "Previous" and "Next" buttons which have viewBox links attached you can make a simple linear slideshow – or you could add more links to let you jump directly to any other part of the file.

Of course you're also free to mix XLink-based links with those created via JavaScript, picking the best tool for the job. One thing you can do with JS which isn't possible with the simpler form, is to provide additional logic to determine the target location. You might change to different URLs based on the time of day, or prompt the user for some additional information that is then encoded into the URL. Consider a web-based storybook, for example, in which XLink is used to move between the pages, but JS provides extra interactivity when elements are clicked on, or hovered over.

Mark uses Inkscape to create three webcomics, 'The Greys', 'Monsters, Inked' and 'Elvie', which can all be found at http://www.peppertop.com/

Inkscape - Part 97

There's some big news in the Inkscape world this month. Since the previous instalment, a couple of new Inkscape versions have become available: 0.92.5 and, after 17 years of development, version 1.0. The former mainly fixes some Windows issues, but does have a few general bug and crash fixes that will benefit Linux users. Perhaps the most significant change is that Inkscape extensions are now compatible with Python 3. Given that Python 2 formally reached its end-of-life at the start of this year, this is a welcome step forward along a path that many projects will need to take, as Linux distros begin to reduce their support for the legacy Python packages.

The full release notes for 0.92.5 can be found here: https://inkscape.org/release/inkscape-0.92.5/

Of more interest to this column is version 1.0. As I've mentioned previously, it's my intention to cover the new features and changes in this milestone release, starting this month. But the first feature I'm going to mention – as much to get it out of the way, as anything else – might seem a little odd for this Ubuntu-focused magazine: I'm going to talk about version 1.0 on MacOS, which represents one of the biggest changes in the history of the Inkscape project.

Although earlier releases of Inkscape had MacOS versions, they were essentially just recompilations of the Linux version. They behaved in exactly the same manner as the Linux program, right down to the keyboard shortcuts. Most importantly, they still required an X server for their graphical output. Since MacOS doesn't natively use X for its display, this meant installing another application, and understanding the relationship between operating system, X server, and Inkscape.

For several years, Apple were proud of the Unix underpinnings of OSX, and shipped their own X server, perhaps as a way to bolster the number of applications available for their fledgling new OS. Over time, this became less of a selling point, and the X server saw little love and attention. Since 2012, Apple no longer maintains their own X server, but does contribute to the Open Source XQuartz project. For the past few years, therefore, it has been necessary to install XQuartz in order to use Inkscape on MacOS. Although this works reasonably well, this combination does have a few idiosyncrasies, particularly when used on a Mac with multiple monitors. So much so, in fact, that I ended up writing my own shell script to rescue me from the "disappearing dialog" issue that plagues this setup (http://www.peppertop.com/blog/?p=1554).

With 1.0, however, Inkscape will be a native MacOS application which doesn't require an X server. With this change also comes better support for the system clipboard, and keyboard shortcuts that more closely match the standards for the OS. Perhaps most importantly, it's compiled as a 64-bit application, whereas earlier versions were 32-bit. Although this latter change doesn't really have much of a bearing on Inkscape itself, it's vital to note that the latest release of MacOS, "Catalina" (10.15), supports only 64-bit programs, so if you want to run Inkscape on that version of the OS, you have no choice but to use version 1.0. Note that it's currently described as a "Preview", which suggests that there are probably a few issues to iron out still, but in my testing, it's proven to be pretty stable.

A native implementation makes Inkscape much easier for Mac users to install and use – which will hopefully boost its user base. As MacOS has a reputation as being an OS for "creatives", Inkscape perhaps faces more competition on that platform than any other. Compared with most other vector graphics programs on OSX, it is Free Software, as well as free software. Whether that will be enough for it to carve out its own niche in the market remains to be

seen, but it's certainly a huge step forward by the developers that should be applauded.

If you're on OSX or Windows, you can download Inkscape 1.0 via links on this page: https://inkscape.org/release/inkscape-1.0/

You'll also find options for Linux downloads there, as an AppImage file or a Snap package. In practice, however, if you're using Ubuntu or a derivative, the easiest way to get the new version is probably using a Snap installed via the command-line. Just run the following command in a terminal:

```
sudo snap install inkscape
```

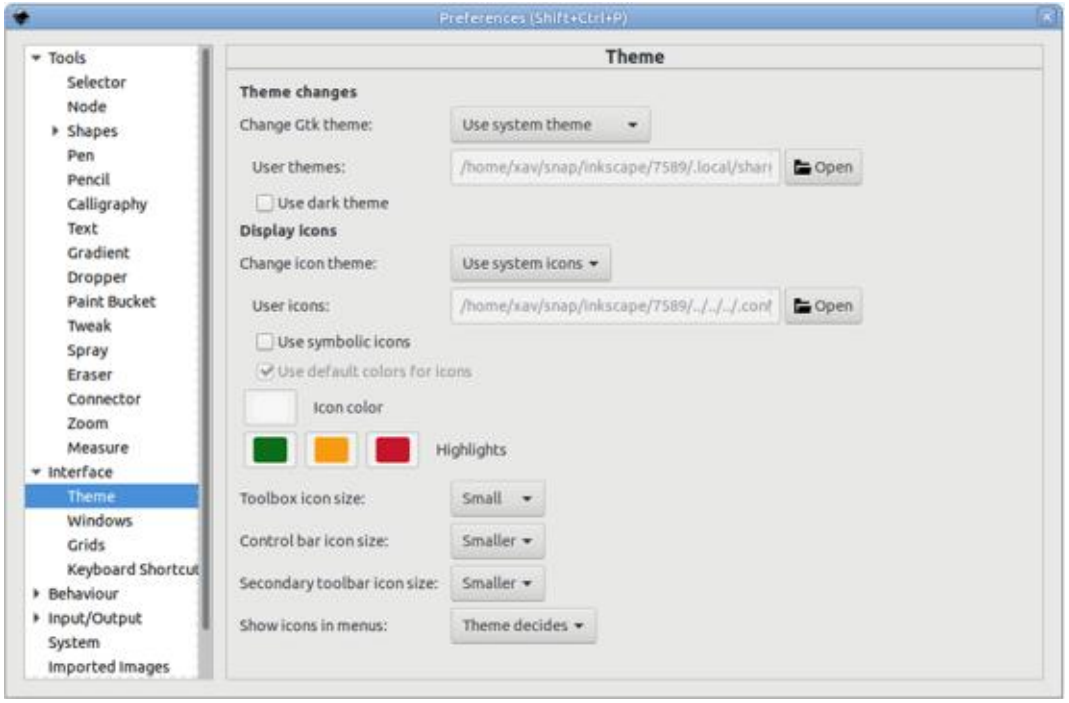
One advantage of this approach is that the Snap-based version will be installed alongside the deb package version from the Ubuntu repositories, so you can try out the new release without having to relinquish the 0.92.x version you might be using currently. You will end up with two Inkscape entries in your menu, but surely the developers would have ensured they're easy to distinguish from each other, wouldn't they? Here's what the entries on my menu look like on Ubuntu Mate, together with a version blown up to double size so you can more clearly see the difference between the icons:

The first entry is the old 0.92.x version, and has a crisper icon with a slightly blue tinge on the right. The second is 1.0, which has a shadow around it, giving a softer look, and no blue on the image. If possible, I recommend renaming one of the menu entries for clarity. The exact way to do this will vary across distributions; on Ubuntu Mate you can run the 'mozo' utility to edit your menu structure (also available by a right-click on the top-level menu button in the panel).

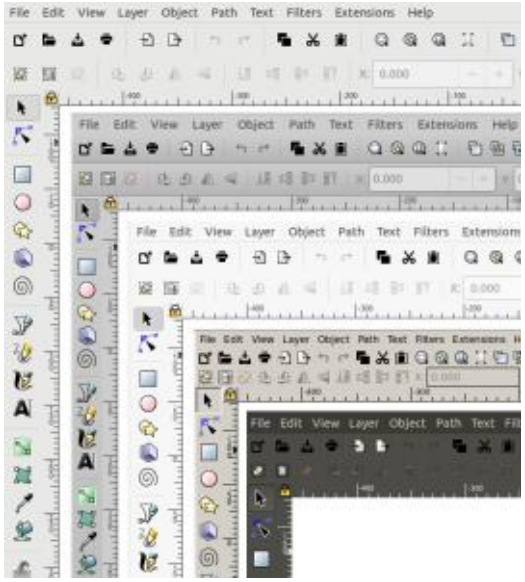
With the 1.0 release installed, it's time to begin playing with the new features. As you do so, please do consider reporting any issues you encounter to the Inkscape bug tracker (https://inkscape.org/report). This is a significant release, and there are bound to be some issues in it. Reporting them will help to improve the quality of the subsequent point releases.

The new release updates the user interface toolkit to GTK3, which has resulted in a slightly different look and feel, but also in significant improvements for users of modern "retina" or "HiDPI" screens. It also brings greatly enhanced theme support, to the extent that there is a new, dedicated "Theme" panel in the "Interface" section of the Inkscape preferences dialog.

The first menu in this panel lets you select from whatever GTK3 themes are available on your machine. On my box, this was a reasonably extensive list which included a number of dark themes, if that's your thing. Selecting an option from this menu updates the Inkscape UI immediately, so it's easy to audition the various styles that are available. Leaving it set to "Use system theme" does what you might expect – although the UI doesn't immediately update if you



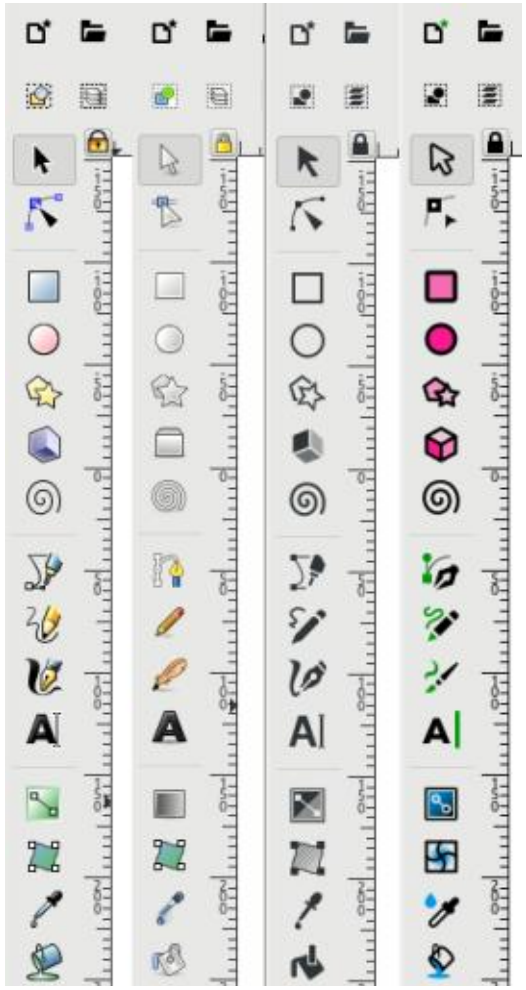
change the system theme via your OS, and will only fall in line after restarting Inkscape. Here's a representative sample of some of the available themes:



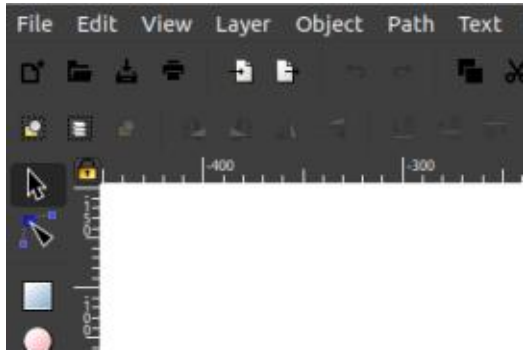
The "Use dark theme" checkbox in the dialog seems a little redundant to me. As far as I can tell, if you select a named entry from the popup, and there's a corresponding "-dark" theme, this checkbox lets you easily switch between the two. However, since all the different versions are listed in the popup, including the dark ones, there's no need for this checkbox anyway. Worse still, some themes have three different entries: in the case of the "Yaru" theme, for example, there is "Yaru", "Yaru-light" (those two appear to be the same), and "Yaru-dark". This checkbox would make more sense if the suffixed versions were suppressed in the menu. You would then select a theme first (e.g. "Yaru") before using the checkbox to toggle between the "-light" and "-dark" variants.

The GTK themes only affect the UI widgets – buttons, scrollbars, text entry boxes and so on – not the icons. The second section in the preferences panel deals with those. A popup lets you select the icon theme you want to use. In the case of my test installation there are four options: "hicolor", "multicolor", "Tango" and "Use system icons". I'm not sure why the first two aren't deserving of capital letters at the start, but that's a minor niggle. Of those choices, three of them result in the same appearance, with only "Tango" looking any different, at least in the default configuration.

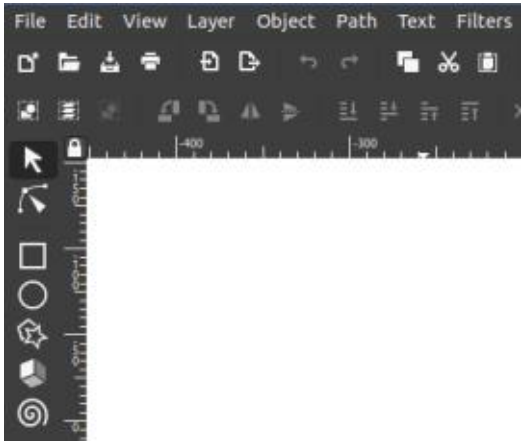
There are actually two other icon styles available, though the UI for selecting them is not entirely straightforward. If you select either the "hicolor" or "multicolor" themes, you will then gain access to a checkbox labelled "Use symbolic icons". With this checked, each theme renders flat icons with little or no shading, and either no additional color (ironically, this is the "hicolor" option), or specific highlight colors ("multicolor"). This screenshot shows all four options, from left to right: hicolor/multicolor (non-symbolic), Tango, hicolor symbolic, multicolor symbolic.



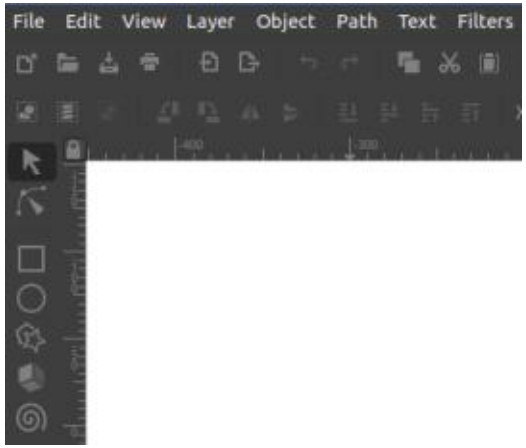
My personal preference is for a light theme with colorful icons. I'm of the view that icons with distinctive colors as well as shapes probably require a little less cognitive effort to recognise, leading to a UI that's easier, and therefore faster to navigate with less conscious thought required. As the work I do in Inkscape generally ends up being printed on white sheets of paper, I usually use a white background in the drawing window. A light theme therefore leads to a less jarring difference between the light color of the canvas and the UI around it. But I know that some people do love their dark themes, and it's for these users that I think the symbolic icons come into their own. Looking at the top-left corner of the Inkscape window with the default icons but a dark theme (Yaru-dark) shows the problem with using the default non-symbolic themes:



Notice how jarring the colored icons are against the darkness of the toolbars. Conversely, there are some symbolic icons visible even with this icon set, which are too dark and difficult to make out (I've reported this as an issue, and the feedback is that it appears to be a packaging problem with the Snap). Switching to a symbolic theme makes the icons a lot more consistent:



Although they're consistent, they're also a bit too bright. By un-ticking the "Use default colors for icons" checkbox, however, you can use the color buttons below to select the main icon color, as well as the highlight colors (for the multicolor theme). This makes it possible to knock the bright icons back to a darker grey – or even a color tint, should you wish.



In the case of the multicolor theme, the three highlight colors are used in the toolbox icons as a means of grouping similar sets of tools together. Once again, this addition of color aids in reducing the cognitive effort required to identify an icon. What I'd really like is the ability to set each base color or highlight on a per-icon basis, so that I could make the tools I regularly use stand out more than those that see only occasional deployment. Perhaps in a future version.

That's enough for the stylistic changes in the UI. Next month, we'll start a deeper dive into the new features and additions that have taken place in the actual drawing tools. If you can't wait, and want a quick preview of some of the highlights of this release, take a look at the highly professional release video the project has posted: https://inkscape.org/release-video-1-0/

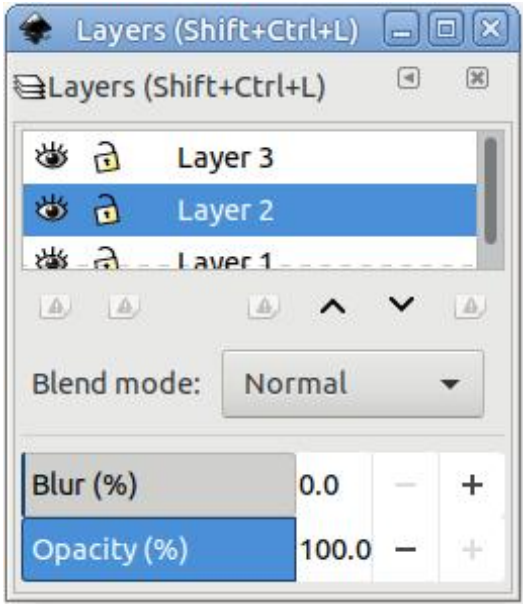
Mark uses Inkscape to create three webcomics, 'The Greys', 'Monsters, Inked' and 'Elvie', which can all be found at http://www.peppertop.com/

Inkscape - Part 98

Last time we began looking at the new features in Inkscape 1.0, and I suggested that you could install the snap version of the package in order to let it live alongside the deb package version of 0.92.x you may be currently using. It turns out that this was both good advice and bad.

The reason it's a good option is because, as I feared, version 1.0 is proving to contain a number of bugs and regressions that range from slightly annoying to full-scale showstoppers, depending on what you use Inkscape for. In that respect, the ability to still switch back to 0.92 makes using the snap a definite advantage.

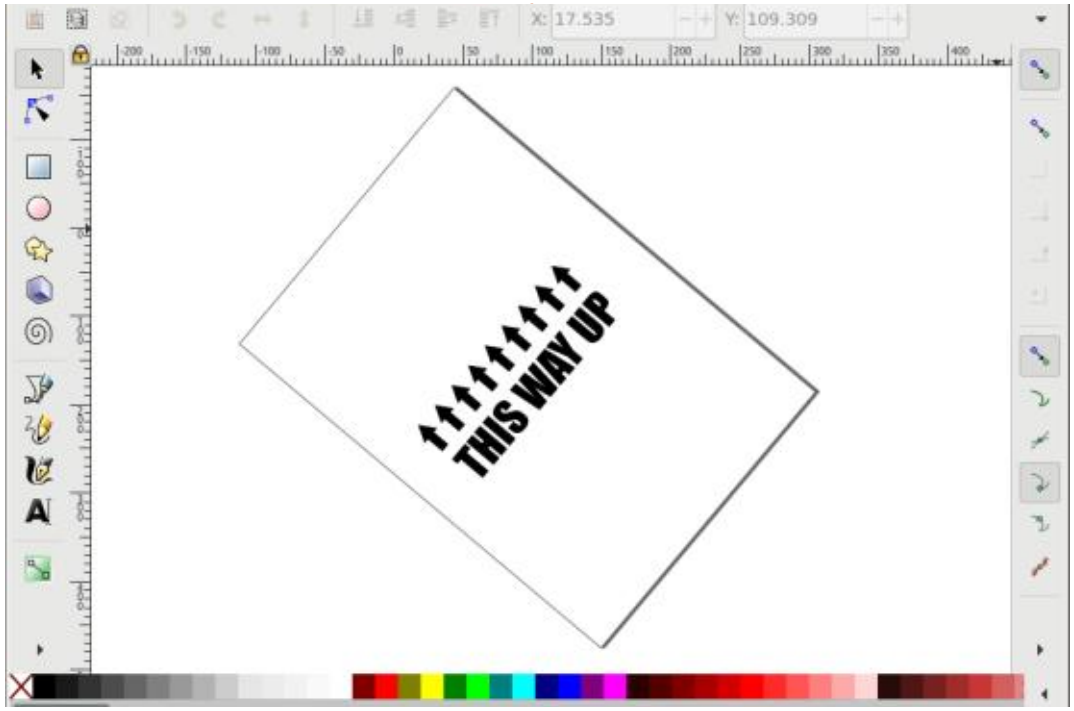
The bad news, however, is that there are some packaging issues with the snap which result in incorrect and missing icons. I noted this problem last month, with regard to some symbolic icons appearing in the toolbar when using a non-symbolic icon theme. Since then, however, I've noticed (and reported) several icons that are completely missing from key parts of the UI. For example, here's the Layers dialog – something that you're likely to use very frequently – missing the icons for adding and deleting layers, and those for moving a layer to the top or bottom of the stack:



These same icons are used (and therefore absent) in various other parts of the user interface, and there are other missing icons too. Fortunately, the buttons still operate as they should, and the tooltips work, so you can still drive the software. Familiarity with older versions helps, but if you're new to Inkscape then the 1.0 snap, as it currently stands, may well lead to confusion.

You might think that switching to a different icon theme would help, but all you get is a different 'missing image' icon. Switch to the symbolic icons, however, and they are all present – which at least offers something of a workaround for now, if you don't mind that aesthetic.

There is another option: there's a PPA available for the 1.0 release which does not suffer from this problem (https://launchpad.net/~inkscape.dev/+archive/ubuntu/stable). By installing this, however, you'll replace the deb packaged 0.92.x version if you have that installed. This is the approach I'll be taking for the immediate future, so that I can produce screenshots for this series that use the same icons as those in earlier instalments – however I will be running it in a virtual machine, as I'll be sticking



with 0.92 to produce my comic strips until the 1.x series matures a little.

There's more to say about the UI changes in Inkscape 1.0, but, right now, you're probably itching to see what new features there are to help you when actually drawing something.

Let's start with changes to the canvas – the entire drawing area within the Inkscape window – beginning with rotation.

Particularly when drawing freehand, it's useful to be able to rotate the canvas to better suit the range of motion in your arm. Think about using a graphics tablet: it's a lot easier to move the stylus from left to right rather than up and down. Of course this doesn't mean strictly horizontal lines, since your arm traces out an arc. But the general principle remains, which can result in artists trying to perform gymnastics to rotate their tablets – and their heads – in order to improve their accuracy for awkwardly angled lines.

With 1.0, it's now possible to rotate the canvas itself, so you can keep the tablet, and your body, in a more comfortable position. If you're using a mouse with a wheel, then you can hold Ctrl and Shift while rotating the wheel. Alternatively, hold Ctrl while you click-and-drag with the middle mouse button (which may also be the wheel). The latter is especially useful for tablet users, for whom the middle mouse button is often duplicated as a button on the body of the stylus.

Some users have reported that they find it too easy to accidentally trigger rotation when they actually intended to pan the canvas. Although there's no way to disable rotation completely in this release, it is possible to assign a keyboard shortcut to reset the rotation quickly if you accidentally trigger it. Within the keyboard shortcuts pane of the Inkscape Preferences dialog you can also set shortcuts to rotate the canvas clockwise and counter-clockwise, though none of these functions have default shortcuts assigned.

If you do rotate the canvas, and haven't assigned a keyboard shortcut for resetting it, you still have a couple of options available for returning to the normal orientation. The obvious option is simply to rotate in the opposite direction but this approach may prove difficult to do precisely. Better is to use a new option in the View menu: View > Canvas Orientation > Reset Rotation.

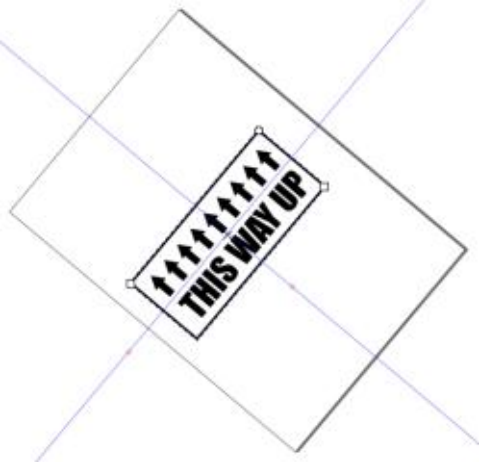
If you want to precisely set the rotation amount, there's an additional field at the right of the status bar, alongside the previous one for the zoom value. You can, of course, type a value into this field directly. But you can also scroll the mouse wheel over the numbers to change the values in 1° increments – or click the +/- buttons to do the same.

In common with most input boxes in the Inkscape UI, there's a context menu that you can access with a right-click on the field, offering a few standard values, this time in 45° increments. This is another useful way to reset the rotation back to zero.

It's all very well being able to rotate the canvas to whatever amount you want, but what does that actually get you as an artist? Unfortunately, the answer is 'not much'. As I mentioned at the outset, this is particularly useful for freehand work, using the Bezier, pencil or calligraphy tools. But if you want to use Inkscape's other features, it has a lot less to offer.

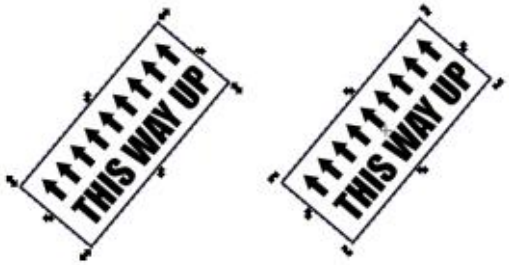
I would have thought that the point of being able to rotate the canvas is that it gives you a way to set an angled baseline against which to create other objects. If you need to create a few lines of text, perfectly positioned at an angle of 26.35° then you would surely just punch that value into the field at the right of the status bar, then plough on with the text tool. Except it doesn't work like that.

Rotating the canvas just rotates your view of the canvas. Text, rectangles and even guides are still oriented to the page, not to the



new view. Here's my rotated canvas with a rectangle drawn, and a pair of guides dragged out after it had already been rotated:

It gets even odder when you select objects. The handles for the selection box maintain their "normal" orientation – so the scale handle for the top-right of the box still points diagonally from SW to NE, for example. But with my page rotated somewhere near 45°, it results in arrows that are distinctly misleading, both in resize mode and rotate/skew mode:



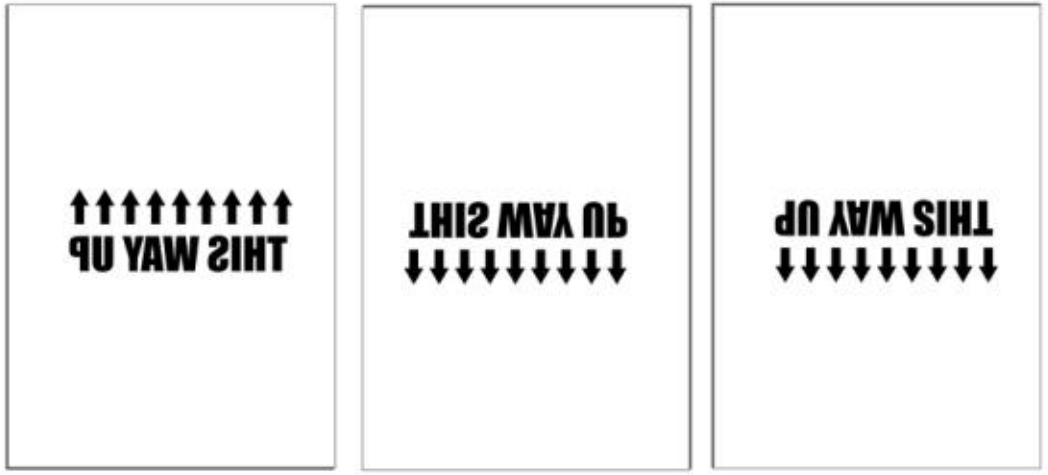
What would happen if you dragged the vertical arrows in the first example? You might think it would scale the content vertically as you look at it. But instead it scales the objects vertically along the axis of the page, not the axis of the screen.

Canvas rotation feels like a missed opportunity to me. Making the other tools, and guides, operate relative to the screen, not the page, would result in some very powerful workflows. Instead it's a helpful feature for freehand drawing, but not a lot else.

Another new feature that's closely related to canvas rotation is the ability to flip the canvas vertically and horizontally – not to be confused with flipping individual objects via the toolbar buttons or the H and V keys. The canvas flipping options can be accessed from the View > Canvas Orientation menu, or you can add keyboard shortcuts for them from the Inkscape Preferences dialog (none are assigned by default). They behave pretty much as you would expect, flipping the canvas view either left-to-right, top-to-bottom, or both (which is equivalent to rotating through 180°). Note that the drop shadow on the nominal bottom-right page border will give you an indication of the current state (if you have it visible, of course).

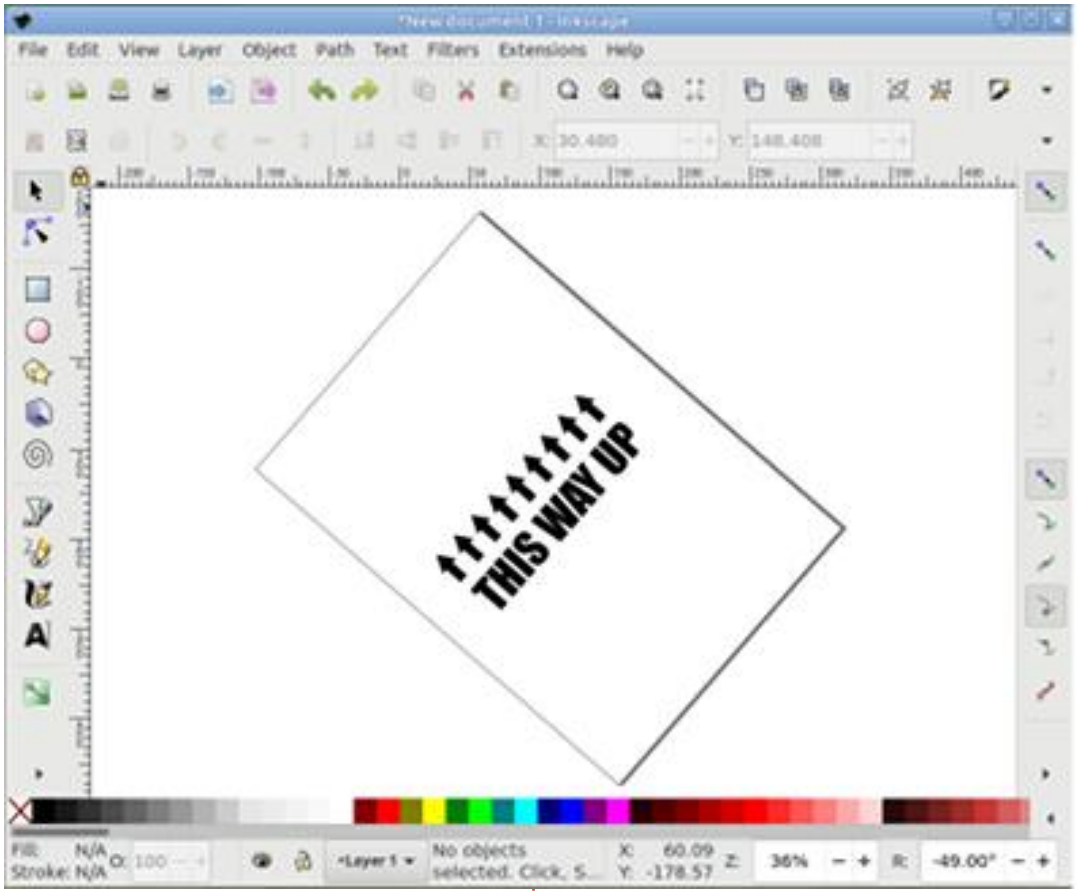
Flipping the canvas is a technique often used by digital artists to get a different view of their work. Sometimes issues in layout or perspective are more obvious in one orientation or another. Simply by making text less legible it can help to expose general design problems without those pesky words vying for your brain's attention at the same time.

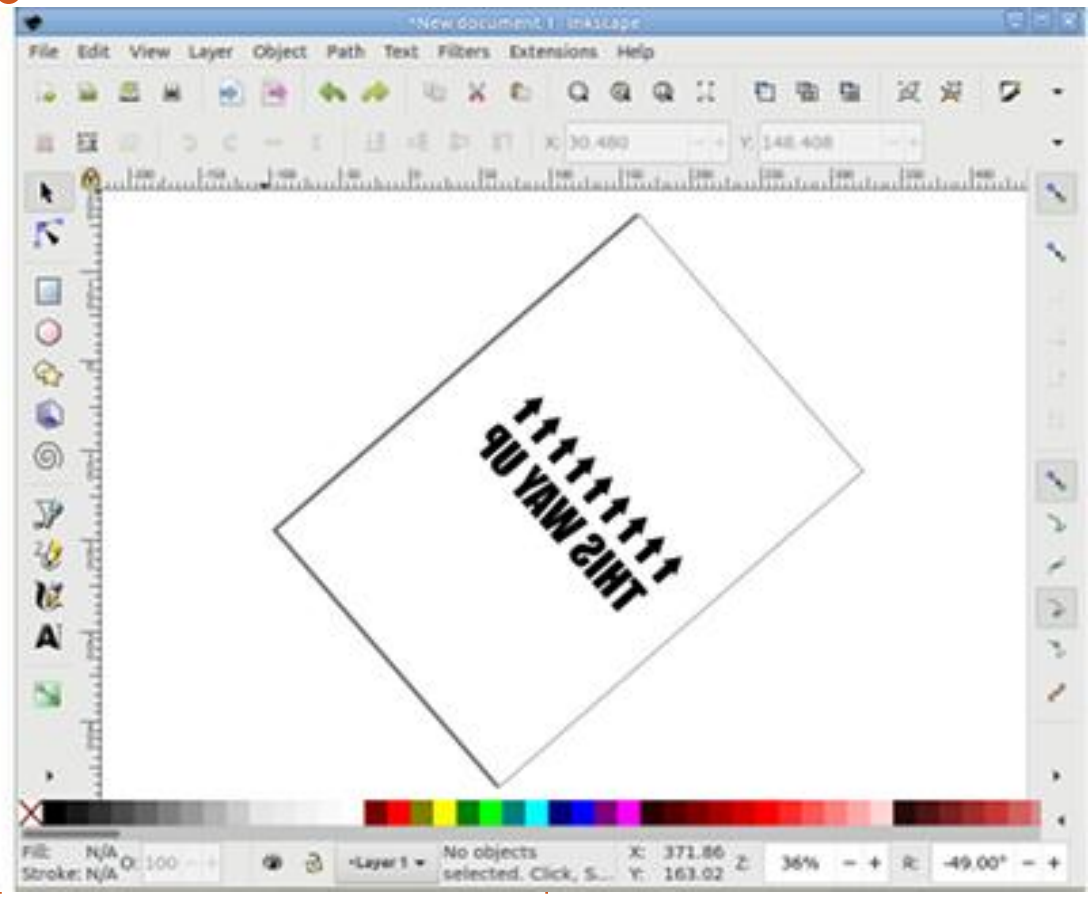
Resetting a flipped canvas is done by flipping it a second time in whichever direction(s) are required. Unless you've set keyboard shortcuts, this potentially means two trips to a nested sub-menu. I'd prefer to see the "Reset rotation" menu entry replaced with a more general "Reset view" option that would not only set the rotation back to zero, but also turn off any horizontal or vertical flipping.



You may be wondering what happens if you apply both canvas rotation and flipping at the same time? The answer is that the rotation is applied first, then the whole viewport is flipped. This is arguably the better ordering for these transformations, as the display switches between two clearly distinct orientations – probably more useful for spotting design problems. For comparison, here's my test page, first rotated and then rotated and flipped horizontally (next page).

One problem that affects all Inkscape users at some time, and novices in particular, is losing your place in a drawing. You've zoomed in, then panned a little too far, and suddenly you're looking at a plain white section of the window with no idea which direction you need to go in to return to your work. With the addition of flipping and

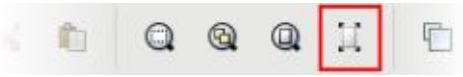




rotating, there's even more likelihood that you may get lost in your own drawing at some time.

Historically you could deal with this situation by zooming out until the canvas was visible in the window, then zooming back in. Or you could use the zoom shortcuts – such as zoom to page – to get back to a known state. But these approaches change your zoom level, which causes Inkscape to re-render the content, causing a real slowdown on some drawings.

A new addition to the zoom buttons in the toolbar for 1.0 is "Center page in window". Clicking this, or using the keyboard shortcut (default: Ctrl-4), pans the content of the Inkscape window in order to position the center of the page at the center of the window, without changing the zoom level. This works based on the nominal page size set in the document properties, regardless of whether or not the page border is visible.



Hopefully you'll find at least one of these new features will be helpful to you. But that's not all for the new canvas and view features in 1.0, so next month we'll continue to look at a few other changes and additions in this area.

Mark uses Inkscape to create three webcomics, 'The Greys', 'Monsters, Inked' and 'Elvie', which can all be found at http://www.peppertop.com/

Inkscape - Part 99

This month I'll be continuing to look at the new view features in Inkscape v1.0, beginning with an addition to the View > Display Mode menu. If you're a long-time reader of this column, then not only do you deserve some kind of award (perhaps an Inkscape-drawn certificate), but you'll also be familiar with the "Outline" display mode.

Known informally as the "where's my invisible stuff" mode, this feature renders every element as a simple outline, regardless of its usual fill and stroke settings. It's invaluable for those times when you've created an otherwise invisible object – whether by accident or intent (there are some good reasons why you might want to). Typically, this is an early mistake by new users who inadvertently clear both the fill and stroke, make the colors transparent, reduce the opacity on the whole object, or go a bit crazy with the blur slider. In these cases, View > Display Mode > Outline lets you see, and select, your invisible object, whether to delete it or try to wrangle it back into a visible state.

The new view mode is related, but different. Whereas Outline mode renders everything as outlines, losing patterns, fills and strokes in the process, the new "Visible Hairlines" feature just makes really thin strokes a bit thicker, but otherwise displays most other objects and properties the same as usual. An example will probably help to explain, so take a look at these five stars:

"Five stars?" you say. "But I can only see two. Maybe three or four if I zoom right in and squint a bit."

There are, indeed, five stars. None of them are hidden. None have any transparency or reduced opacity applied. None are filtered. Yet some are definitely easier to see than others – and one is not visible at all. What's going on here? Perhaps if I describe each star you might realise what's happening:
Top left: 1px black stroke.
Top right: 64px black stroke.
Bottom left: 64px light gray stroke.
Bottom middle: 0.1px black stroke.
Bottom right: Red fill.

The visible stars on the right are easy enough to explain – one has a thick stroke, one has a fill. The bottom left also has a thick stroke, but in a light color that makes it hard to see. The top left is possibly just about visible: these are large stars, and the page is zoomed out, so that a 1px stroke is just on the edge of Inkscape's display capabilities. If you go much thinner – as I have done with the 0.1px stroke on the remaining star – Inkscape just doesn't render it at all.

Let's see how these look in Outline mode.

Here we see the problem with Outline view. Yes, we can now see all the stars, which is great for finding invisible objects. But they all look the same, regardless of their original styles.

Compare this with View > Display Mode > Visible Hairlines:

Although it may not be clear in the magazine, the top left star is ever so slightly thicker. Our previously invisible star is also rendered with the same thickness.

The two stars on the right just appear as they usually would – as does the star at the bottom left, which is still difficult to see.

This view mode is intended to help if you are designing an image for use with a laser cutter or similar device. These machines often require paths to have a purely nominal thickness – 0.1mm is a typical value – which can easily result in them disappearing from view when you zoom out. By switching to Visible Hairlines mode, you ensure that Inkscape will always display strokes at a thickness that is enough to keep them visible. It doesn't change the actual widths in the SVG file, so your laser cutter won't complain, but does mean that objects no longer vanish while you work.

Any stroke that is already above the minimal limit will be rendered as usual. That's why the thicker strokes on the top right and bottom left stars are unaffected. As you can see with the bottom left, however, that can still lead to barely visible shapes, so Outline view still has its place. As the bottom right star shows, fills and other properties are also rendered as normal in this view. Filters don't fare very well, but as they are purely visual effects that don't alter the geometric information that a laser cutter might use, they're not generally used in the sort of situations that this mode is intended to address.

Most users won't need this mode. If you just want to find an invisible object, Outline mode is still your best bet. But if you do have to work with exceptionally thin strokes, and find they vanish when you zoom out, this mode is just the thing for you.

If you are on the hunt for invisible objects, or those hidden behind other items, there are a couple of other new features hanging around on the View menu: View > Split View Mode and View > XRay Mode. They're a little like dynamic versions of the Outline view mode, giving you the best of both worlds – both outline and full display, at the same time. They should also both be on the Display Mode submenu, in my opinion, but probably live on the main menu to make them more discoverable. Let's look at Split View Mode first.

This image is made up of four stars, identical except for their size. I created the smallest one first, and set its opacity to 25%. Then I duplicated it and resized to create the second one; then duplicated and resized the second to make the third; same for the third to the fourth.

My question to you is this: what's the easiest way to select the smallest star? Due to the order in which I created them, it's at the bottom of the z-stack, so a simple click won't do the job. Here are some methods that would work in this particular case:

- Careful rubber band selection.
- Hold Alt (or Super-Alt on most Linux boxes) and repeatedly click in the same spot to select below the current object.
- Reorder the stack manually before selecting.
- Reorder the stack using Extensions > Arrange > Restack before selecting.
- Switch to Outline view mode, then select.

The last option is arguably the most practical. It doesn't require you to change the arrangement of items in your drawing, and is probably less likely than the others to result in the wrong thing being selected. But it's a pain to turn Outline mode on and off via the menu, and you may not use it often enough to warrant learning more keyboard shortcuts.

What would be really handy is a way to get a temporary outline view of the drawing, but then switch back to normal view simply and intuitively. With Split View Mode that's almost what we've got. Almost.

You can activate the new mode, via the View > Split View Mode menu. The default keyboard shortcut is Ctrl-6, but this is 6 on the main keyboard, not the numeric keypad. I find this a little odd, given that the default

shortcut for cycling through the display modes uses Ctrl with 5 on the numeric keypad. However you trigger it, enabling this mode will immediately split the workspace in two, with the left side showing the normal view of your document, and the right side showing the outline view.

A large circular handle at the middle of the screen can be dragged to move the split point, allowing more or less of the view to be displayed in outline mode. Four triangles within the control let you switch the orientation of the split between horizontal and vertical, and determine which side of the split should show the outline view. Both sides of the display work as normal, so you are free to select items in the outline view then manipulate them in the normal view, or vice versa. Clicks and drags on the splitter control are not propagated through to the objects below so you can, for example, make a selection in outline view then adjust the split to give you more of a normal view without the objects becoming deselected as you do so.

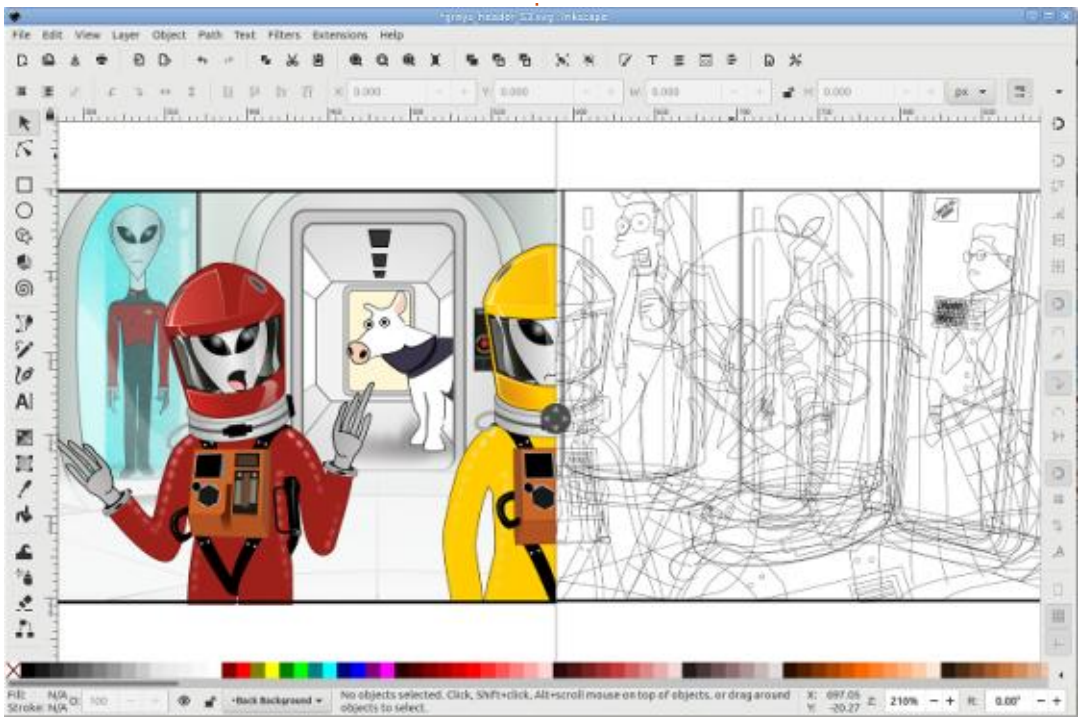
On the surface, this looks like a great addition for working with complex documents. Just enable Split View Mode then slide the splitter to one side to give you a normal view most of the time, swinging it back into play for tricky selections, before swiping it away to the side again, ready for redeployment in an instant. Except it doesn't work like that.

If you drag the handle too close to the edge of the workspace, this mode gets deactivated, requiring another trip to the View menu, or hitting the keyboard shortcut, to switch it on again. But there's no visual indication of where "too close" begins. And it's deactivated even if you're still in the process of dragging, without the courtesy of at least waiting for you to release the mouse button! The result is that it's too tricky to leave this mode enabled at all times, just swiping the splitter in and out of view as needed, because you're bound to swipe it too far at some point and turn the feature off.

Worse still, moving the big handle towards any edge has the same effect, regardless of the splitter orientation. With a vertical splitter, separating the two halves to the left and right, you might be tempted to move the handle up or down to get it out of the way of your content. But even though you're not moving the splitter itself, drag it a pixel too high or low and the whole mode turns off, requiring you to start at the menu again.

This mode also doesn't store the split state between invocations. Every time you enable it you'll be put directly into the same 50/50 view with the outline on the right. It doesn't matter if you previously switched to 70/30 with the outline at the bottom, or any other combination. Once this mode is turned off, and then back on again, it drops you straight to the default. And given how easy it is to accidentally turn it off, it makes working with this mode more of a pain than a pleasure. It does make for nice screenshots when demonstrating the density of objects in a complex drawing though.

View > Xray Mode is a closely related, but slightly different feature. The keyboard shortcut for this is Alt-6, by default, but once again this has to be the 6 on the top row of keys, not on the numeric keypad. Switching this on enables outline view in a circle immediately

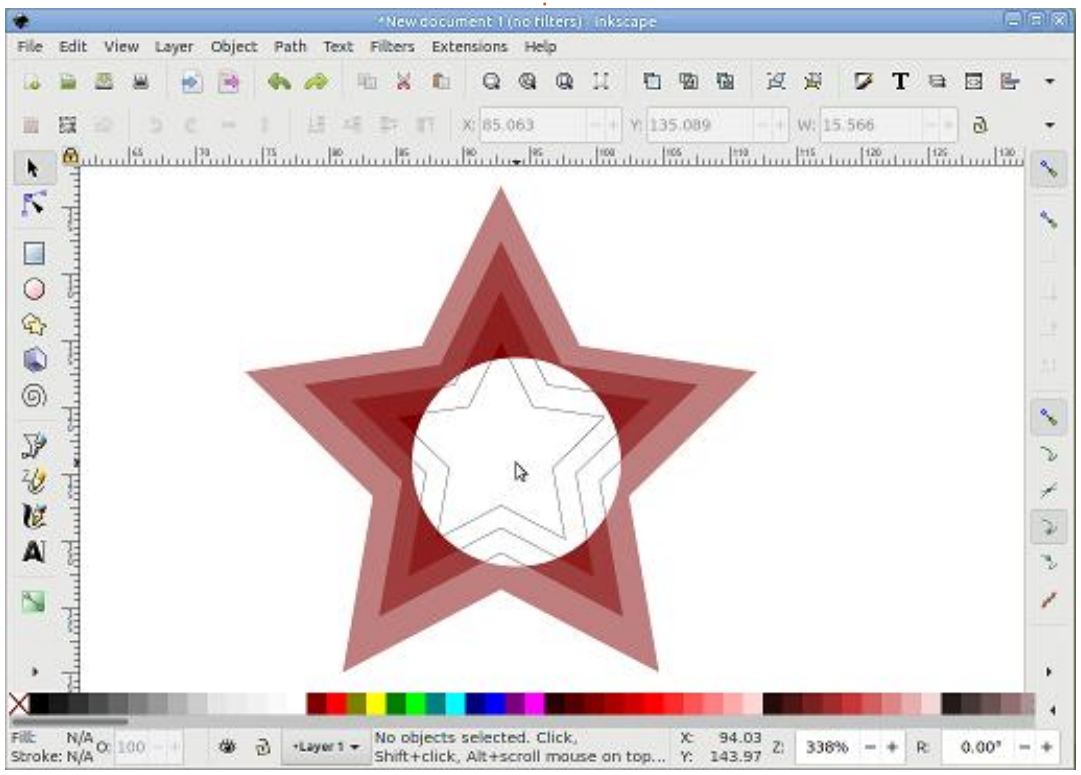


surrounding the mouse pointer. Like an X-ray, it cuts through the body of your drawing to show you the bones that support it. But it's more like a tightly controlled "X-ray beam" from a pulp comic book than the large format plates that are routinely hung from wall-mounted light boxes in medical dramas.

As with Split View Mode, you can still make selections and perform other tasks with this mode activated. There's no way to change the size or shape of the beam, and no way to 'leave it' at a particular place while you use the mouse elsewhere. To my mind this mode is a fun distraction, but ultimately not very useful for day-to-day work.

If you struggle to select objects in busy drawings, then it might be worth changing the keyboard shortcut to make it easier to toggle on and off. What would be nice is a 'hold-to-activate' key, like the 'Q' key for the quick-zoom function, which would allow you to press and hold a key for long enough to make a tricky selection, but would return you to the normal view as soon as you release it. It would certainly make for a better use of the 'X' key than as a shortcut for the largely useless 3D box tool. I have, of course, filed a feature request for this – as well as one for improvements to the Split View mode.

That's it for the new view features in v1.0. Next month will be the 100th article in this series, so I'll be celebrating this milestone by… continuing to bring you details of some of the other new features in the latest Inkscape release.



Mark uses Inkscape to create three webcomics, 'The Greys', 'Monsters, Inked' and 'Elvie', which can all be found at http://www.peppertop.com/

Inkscape - Part 100

Written by Mark Crutch

Amazingly, this is the centenary article in this series! That's not bad given that my original plan was for maybe half a dozen short pieces as a brief introduction to Inkscape. I doubt there are many people who have read all of the articles, but even if you just dropped into one or two as reference material for a specific feature, I hope you've found them useful.

This month, I'll be continuing to look at Inkscape v1.0. Although there are a lot of exciting new things to cover – and I promise to get to them in the coming months – for now I'm going to return my focus to some of the problems that have found their way into the new release. It's not that I want to be a pessimist, I'd just prefer to get the issues out of the way as quickly as I can.

The regressions that most users are likely to stumble across are largely related to path operations on text objects. Manipulating text to produce a graphical effect is a common task for Inkscape users, when creating a logo or just trying to make the text on an image look more interesting. There's little you can do with a pure text object, apart from the usual rotate, scale and skew. Filters can produce some amazing results – whilst still keeping the text editable – but are difficult to work with, and are best used to provide texture or shadows rather than for fundamentally changing the letter shapes. At some point in your graphical adventures, you're bound to find yourself converting text to a path in order to manipulate it further.

Converting Text to a Path

Logos and styled text form the basis of a huge number of Inkscape tutorials, almost all of which require the user to convert their text to a path at some point, via the Path > Object to Path menu option. Depending on the age of the tutorial, and the version of Inkscape it targets, this can lead to very different outcomes.

For most elements in Inkscape the result of Object to Path has always remained the same. A rectangle becomes a path. An ellipse becomes a path. Spirals and polygons become paths. It may sound obvious, but in most cases the Object to Path feature turns your object into a path. With older versions of Inkscape (prior to 0.48) the same was true of text objects: the entire text would be converted into a single complex path. Most tutorials deal with applying an effect or style to the entire text, so creating a single path is just what is wanted. But sometimes you need to be able to work with the individual letters. Splitting the complex path into separate letters is a fiddly task that requires some experience to do easily. So, with 0.48, the developers changed the Object to Path behaviour when applied to text elements: to this day it creates a group of paths, one for each letter.

This change in behaviour immediately made a lot of tutorials slightly obsolete, and led to a persistent trickle of new users to Inkscape support sites posting requests along the lines of "I'm following this tutorial on YouTube and I get as far as converting my text to a path, then the next step doesn't work…". In case you were in any doubt about the longevity of content on the internet, the trickle of users following tutorials written for 0.46 and earlier continues to this day.

The 'obvious' solution to this change is to ungroup the converted text, then join the paths together using Path > Union or Path > Combine. But whether by accident or design, the developers did leave a shortcut in place that allowed for a one-step conversion of text to a single complex path. Rather than use Object to Path, selecting the text object and going straight to Path > Union would have the desired effect. This quickly became the common response to new users following older tutorials, and also became a prominent method in newer tutorials. It's worth noting that Path > Combine, when applied to a text object, also created a single path but it still puts it inside a group, which is arguably a little

useless. Nevertheless, for 0.48 and 0.92 there are four ways to convert text to one or more paths.

Unfortunately, Inkscape 1.0 removes the Path > Union shortcut. To make matters worse, the status bar shows an "error" stating that "one of the objects is not a path, cannot perform boolean operation", even though it does convert the text to paths, albeit in the same way as the Object to Path option.

So now there's a resurgent flow of new users to support sites who are finding that tutorials for 0.48 or 0.92 are also incorrect with the new version. But as there's no longer a one-step solution to this task, the answer is usually to use the three-step approach.

The "3-step" method to convert text to a single path in Inkscape 1.0:

1. *Path > Object to Path*
2. *Ungroup (leaves the paths selected)*
3. *Path > Union or Path > Combine*

Inset and Outset

Related problems appear when using Path > Inset and Path > Outset. In both these cases, Inkscape 0.92 and earlier handled text objects as a special case: the text would automatically be converted to a single path for you before the effect was applied. But with 1.0, the results are not only not what you might expect, but the UI is downright misleading!

With your text object selected, choosing one of these options might be expected to make it thinner (Inset) or thicker (Outset). Earlier versions did this, via the implicit conversion to a path, so although there were some users who were surprised that their text was no longer editable as a text object, for most the experience was in line with expectations. Try the same with 1.0, however, and the status bar says "No paths to inset/outset in the selection". What it doesn't immediately make clear, however, is that rather than failing benignly, leaving your text object untouched, it has also converted your text to a group of paths! So not only does it fail to do what you originally wanted, but it's changed the nature of your object in the process.

Initially it might seem that the quick solution is to apply the inset or outset effect twice: the first converts the text to a group of paths, and the second then applies the desired effect to those paths. But alas, the first operation leaves the group itself selected, not the paths, and applying inset/outset to a group doesn't work, even if that group consists of nothing but paths. Instead, the best solution is to perform the three-step conversion above to convert the text to a single path, and only then apply the inset/outset operation.

Method	Result (0.48 and 0.92)
Path > Object to Path	A group containing one path per letter.
Path > Combine	A group containing a single complex path for the whole text.
Path > Union	A single complex path for the whole text.
Path > Object to Path, then Ungroup, then Combine or Union	A single complex path for the whole text.

Method	Result (1.0)
Path > Object to Path	A group containing one path per letter.
Path > Combine	A group containing a single complex path for the whole text.
Path > Union	An apparent error message in the status bar, but it still converts the text to a group containing one path per letter.
Path > Object to Path, then Ungroup, then Combine or Union	A single complex path for the whole text.

Dynamic Offset

The Path > Dynamic offset operation seems to be particularly misleading when applied to text in version 1.0. In older releases, the behaviour was similar to inset/outset: there would be an implicit conversion to a single path before the real operation was applied. In this case, the operation results in a small diamond-shaped handle that can be dragged to dynamically adjust the inset or outset amount.

Not so in 1.0. Initially, there's no visible change, except that the selection box has lost its handles, and the pointer looks different. That's actually because Inkscape has switched you to the Node tool. The usually helpful status bar is now positively misleading. "Drag to select objects to edit", it begins. So you try dragging across the whole text: no change. You drag over a few of the letters: now the selection box disappears completely.

"… click to edit this object", it continues. So you click on a letter, and are presented with the node handles for the path making up that one letter. Because, yet again, when you first clicked the Dynamic Offset menu entry, your text object was converted into a group of paths, one per letter. You can click on other letters, and in each case the previous one is de-selected, and the clicked one shows its nodes. Shift-clicking selects multiple letters. You can drag the nodes, add them, delete them, and generally work with the Node tool as usual. But there's no handle presented for dynamically adjusting the inset/outset which was, after all, what you were trying to do in the first place.

As with inset/outset, the dynamic offset effect doesn't work when applied to a group of paths. Unlike those operations, however, it also won't work when applied to a selection of several paths – instead dropping you into the Node tool, even on earlier Inkscape releases. This means that there are no shortcuts: you simply can't avoid converting your text to a single path if you want to use the dynamic offset feature on it. It's back to performing the three-step conversion process first.

Linked Offset

The last command in this group, Path > Linked Offset, also fails to work with text in 1.0, but this time the workaround is quite different – it's more complex, but does at least provide some useful functionality in the process. Let's have a quick refresher on linked offsets in 0.92, and why I think they're usually a better option than inset/outset or dynamic offsets when working with text.

If you look at the SVG output for each of these commands, you'll start to understand the differences between them. Inset and outset simply create a new path element that replaces your original object entirely. There's no trace of it left in the file. They're useful when you just want to make another path element a bit fatter or thinner, and don't need to retain the original path in any way. Viewed in this way, you can understand why text (or any other shape) has to be converted to a path for these operations to work.

Dynamic offset is a little different. Yet again a path is created, completely replacing the original object. Like all SVG paths, it contains a "d" attribute which holds the details of the path shape, so that other programs can display it correctly. But this path has some additional Inkscape-specific attributes that are not usually present on a path. The first is "sodipodi:type" with a value of "inkscape:offset", indicating to Inkscape that this path should be treated differently to normal paths.

The second special attribute is "inkscape:original" which holds a copy of the original path that was used to create this offset path. Although the original path has been removed from the drawing, it lives on in this attribute, the content of which can even be copy-pasted into another path object to reconstruct the original shape. The third attribute, "inkscape:radius", holds the size of the offset – i.e. how far you have moved the handle from its original position.

When a path has these attributes set, Inkscape no longer uses the "d" attribute to draw the path. Instead it calculates the new shape based on the original path and the radius (offset) value. Change either of these and you'll see that Inkscape updates the "d" attribute automatically, but this is

just so that the shape appears correctly in other software.

A linked offset also creates a path, but leaves the original object untouched. This path is similar to the dynamic offset case, but adds one more special attribute into the mix, in the form of "inkscape:href". This holds a reference to the original, untouched object.

The new path still contains a "d" attribute, for other software to use. But it also still contains an "inkscape:original" attribute, again holding a path shape. This time, however, the path data here is created by implicitly converting the source object (the one pointed to by inkscape:href) into a path. If you change the source object, Inkscape updates the "inkscape:original" value which, in turn, results in an update to the "d" value.

What does that mean in practice? It means that your original object retains its original type. Stars are still stars, and can be edited with the appropriate tool. Spirals and rectangles, again, are left untouched and can still be edited with their own tools. And yes, text is also left as text, meaning that you can change the font, style and even the content, with the linked offset updating automatically. This alone makes it the best option for offsetting text. You can hide the original or place it off the page if you want, but when you later discover a typo, or need to edit the words, you can make those changes to the original text and know that your offset version will stay in sync.

But not in Inkscape version 1.0.

Yet again in this version trying to use the command just converts the text to a group of paths and switches you to the node tool. You can use the three-step conversion to create a path before you select the menu entry – but that completely defeats the point of using a linked offset rather than a dynamic offset.

Luckily the underlying rendering engine isn't broken. If you present it with a linked offset created in an older version of Inkscape it displays correctly, and the original object still retains its original type and editability. Fortunately, we can reproduce this behaviour entirely in 1.0, but it does involve some use of the XML editor. These are the steps you'll need to follow:

- Select your text object and open the Object Properties dialog via the context menu, or the Ctrl-Shift-O shortcut.
- Copy the ID from the dialog to the clipboard. Close the dialog.
- Create a temporary object. Just about any sort will do, but I usually use a rectangle or ellipse. Making it a distinct color will help to keep track of things later.
- Apply the Path > Linked Offset function to the temporary object.
- With the linked offset selected, open the Edit > XML Editor dialog.
- The linked offset should already be selected in the editor. You should see the extra attributes I described previously.
- Change the inkscape:href attribute: delete most of the content, but leave the "#" in place, then paste the ID from the clipboard immediately after it.
- Press Enter to make the change. You should see your text apparently change to the color of the temporary object. What you're actually seeing is the linked offset positioned on top of your original text.
- The Inkscape UI is a little confused at this point, as the temporary object will still be showing the linked offset handle. Switch to the Selection tool to get things back under control.
- Delete the temporary object.
- Double-click on the linked offset text, or select it and switch to the Node tool and you should see the usual offset handle. Adjust this to change your offset amount.

It seems like a lot of steps, but all we really did was create a linked offset to another object, then adjusted the link to point to our text object instead. Most importantly, our text remains untouched, so can still be edited using the normal text tools, with the linked offset updating automatically.

One final thing to note is that the steps above also work for other shapes. If you want your rectangles to remain as rectangles, or your stars to still be editable with the star tool, you'll need to follow those steps to avoid Inkscape 1.0 automatically converting your source objects into paths.

Reversing Sub-paths

I mentioned this issue back in part 95, when Inkscape 1.0 was still in Beta. Unfortunately the final release still suffers from the same

problem.

With older versions, if you have a complex path (i.e. one with sub-paths), you could select a single node in the sub-path, then use Path > Reverse to reverse just that sub-path. This can be invaluable when working with the nonzero fill-rule, as described in part 95. In Inkscape v1.0 the Path > Reverse option reverses every sub-path, regardless of what was selected.

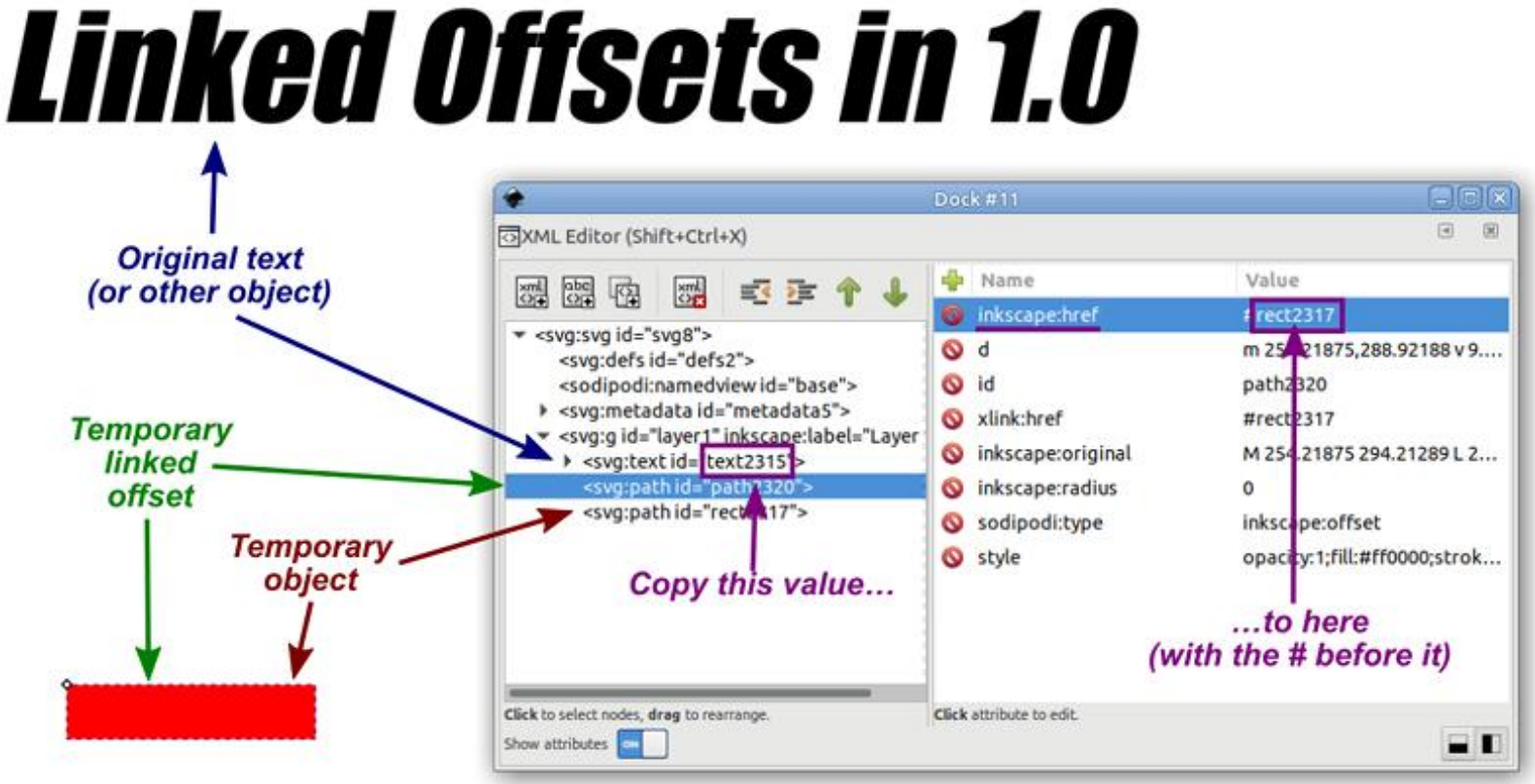
There is only one way to work around this issue, and it's something of a pain on more complex designs. You have to use Path > Break Apart, then reverse the sub-path in question, then select all the constituent paths before using Path > Combine to put them back together into a single complex path.

Final Thoughts

I hadn't really planned to celebrate 100 articles with a text-heavy trawl through bugs and a deep dive into offsets, but these are the sort of problems that will affect lots of users, so warranted spending some time on. It just goes to show that new isn't always better, and makes a strong case for sticking with 0.92 for the time being, especially if you're a new user following some online tutorials.

All these problems have been reported, and I have added extra comments or information on the bug reports where necessary. Hopefully the next release will address some of these issues – many of which look as though they have a common underlying cause.

Next month I'll move back onto some of the good stuff in Inkscape 1.0, hopefully without bumping into any more serious bugs along the way.



Mark uses Inkscape to create three webcomics, 'The Greys', 'Monsters, Inked' and 'Elvie', which can all be found at http://www.peppertop.com/